A Project Report on

A Lightning Man Alive: 3d Game

Submitted in partial fulfillment of award of

BACHELOR OF TECHNOLOGY

Degree
in

COMPUTER SCIENCE & ENGINEERING

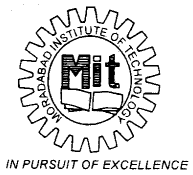
By

SAKSHI GUPTA - 1508210121
MANPREET SINGH - 1508210069
PRANJAL GUPTA - 1508210101
PRAJJWAL BHARDWAJ - 1508210099

(2015-2019)

Mr. Praveen Saini
Assistant Professor

SUPERVISOR



Deptt. of Computer Science & Engineering

Moradabad Institute of Technology
Moradabad (U.P.)

CERTIFICATE

Certified that the Project Report entitled "**The Lightning Man Alive: 3D Game**" submitted by Sakshi Gupta (1508210121), Manpreet Singh (1508210069), Pranjal Gupta (1508210101), Prajjwal Bhardwaj (1508210099) is their own work and has been carried out under our supervision. It is recommended that the candidates may now be evaluated for their project work by the University.

Mr. Praveen Saini
Assistant Professor

Date:

ABSTRACT

The Project consists of a 3D game which is developed by using two software's, namely Blender and Unity 3D max. In the game, there is a 3D environment of the series of 'The Flash. There are two different characters i.e. One Male Character named **"Flash"** and Female character named **"Nora"** who would be playing the game.

Blender is basically used for modeling purpose and Unity is the game engine wherein the scripting and animation for the game is done. Work in the Blender is substantially faster and more interactive with new improvements that help accelerate navigation, selection, and viewport texture baking, especially with dense meshes and scenes with texture maps. Increased productivity comes when working with high-resolution assets required by today's demanding.

Entertainment and design visualization projects. Blender Entertainment creation Suite standard helps us to cheese software for more flexible production pipelines based on project or client requirements create more precise models from real-world references. see the effects of chancing lightning or materials interactively with enhancements, create advanced viewport shader , better and organize data, model using extensive polygonal. subdivision surface and NURBS toolsets within Blender to help create better assets in less time, create more realistic rigid and soft body dynamics, hair, fur,

cloth, fluids and particle simulations; access specialized tools for motion capture acquisition, refinement and retargeting to help us use and reuse motion captured data more efficiently: use the real-time 3D engine and dedicated toolsets for character rigging ,nonlinear animation editing, and on-set acquisition to help increase projects we can tackle; animate more realistic character reactions to external forces create highly detailed organic models experience a more cohesive, efficient pipeline solution and help to increase efficiency with more consistent interfaces that offer unified hotkeys for viewport navigation and certain other tasks. On the other hand, we have the unity game engine in which all the objects of Blender is imported and final scripting is done. The final game is therefore build unity. Unity is an excellent platform for beginners who want to learn more about game development. In addition, it provides developers with online tutorials to begin game development and practice different techniques to build high quality games.

Unity IDE provides an interface for beginners to start the game development. The user interface is implemented in a way where the design phase and backend is separated to reduce the complexity of the development. The animations and visuals are written in C++ for performance. Development in unity is divided into modules. Developers can include audio and video in games; create characters and scenes directly using a visual interface. Unity was launched at Apple's Worldwide Developers Conference to build functions and projects on Mac computers. It was later was introduced for other platforms. Unity performs as a game development environment providing intuitive tools that help you design awesome 3D content. It also provides cross platform publishing, millions of ready-made assets in the Asset Store and an online community For individual developers and studios, Unity's environment reduces the time and

cost allowing you to develop uniquely beautiful interactive games. It provides flexibility to deploy projects on multi-platforms like iOS, Windows and android. Unity game engines offers a vast array of features and a fairly easy to grasp interface. Its bread and butter cross-platform integration, meaning games can be quickly and easily ported onto Windows, making it a great game engine for the development of Windows game.

ACKNOWLEDGEMENT

We express our deepest sense of gratitude towards our guide Mr. Praveen Saini Assistant Professor of CSE Department, and our project committee Mr. Vikas Kumar (H.O.D), Dr. Neelaksh Sheel, Dr. Lal Pratap Verma, Mr. Himanshu Agarwal, Mr. Manish Gupta department of Computer Science and Engineering, Moradabad Institute of Technology, Moradabad for their patience, inspiration, guidance, constant encouragement, moral support, keen interest and valuable suggestion during preparation of this project report.

We would like to express our special thanks and gratitude to all the faculties and non-teaching staff of our CSE Department for giving us their special attention and valuable time. We are highly indebted to Moradabad Institute of Technology for their guidance and supervision.

SAKSHI GUPTA (1508210121)

MANPREET SINGH (1508210069)

PRANJAL GUPTA (1508210101)

PRAJJWAL BHARDWAJ (1508210099)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION TO THE PROJECT

"The Lightning Man Alive" is a 3-D game and created as a major project for Computer Science department at Moradabad Institute of Technology. The aim of this document is to describe the development process, animation effects and the results of this project.

1.1 BACKGROUND

Despite the economic instability and crisis deeply affecting the world, analysts published that the game industry has grown at a rate of 13.3% year over year surprisingly. Even as we type these words, millions of people are playing game in front of their computers. The reason of this growth can be stated that the game industry can appeal any users with different tastes. Some companies realized the opportunities in this field and developed a new technology Blender and Unity game development studio for game developers. The Idea of developing game is not new for us. However, our team consisted of four students, Blender and Unity was a new platform to learn and none of us was familiar with game development. Thus, we had to waive a chance to implement our ideas. Actually, at the beginning of the project we had still no experience in developing a game. But, this time there was a big difference that we were be more determined than ever as everyone knows, there are several types of computer games me first person shooter is one of the best-known game genres. A Subway surf is a game which makes the player feel within the game world; as if the main character has the game camera fixed at his eyes. After the success of doom which is accepted as the first 3D game, many companies took a crack at this type of game. Nowadays the best seller game is a kind of first-person shooters.

1.2 PROJECT OBJECTIVES AND DELIVERABLES

The purpose of the project to design and Implement a 3D game written in C# using Blender and Unity game engine. The project includes a complete level of game with documentation. The level will include everything that should be available in a 3D game. The game will be a single-player game. The team members don't take aim at developing an Instructive game; instead the aim is action, action and more action.

The various Project Deliverables are:

- Project Plan
- Game Design Document
- Test Plan
- Test Report
- Final Report
- Product
- User Manual

1.3 PROJECT ORGANIZATION

Because the team consists of four members, we don't define any specific role for any members. Each member takes responsibilities of each role in each phase to guarantee that the project can continue.

1.4 OVERVIEW OF THE GAME

The project consists of 3D game which developed by using two software's, namely Blender and Unity; Blender basically used for modeling purpose and Unity is the game engine wherein the scripting and animation for the game is done. In this game there is a 3D environment of the series Flash.

There are two characters in this 3-D game i.e. male character named “Flash” and female named “Nora”, both are infinite runners having lightning power with them which makes them to run faster, faster and more faster.

The game is available in two different modes that are Light mode and Dark mode as shown in Fig 1.1 and Fig 1.2. The game is divided into levels and difficulty of the levels get increased as much as you qualify levels or we can say level by level difficulty level will increase. In this game, the task is to collect coins as more as you can in given time limit by crossing hurdles in form of cars and trucks. As you cover more distance, your speed also get increased. It means distance is directly proportional to speed but inversely proportional to time.

Basic formula used: Speed = Distance/Time (i.e. s=d/t).

Fig 1.1 Overview of Game (Dark mode)

Fig 1.2 Overview of Game (Light mode)

1.5 CONFIGURATION OF THE GAME

"The Lightning Man Alive" is an action based 3D infinite running game where the player takes the role of the main character to cross all hurdles and complete the task. The game is inspired from a famous series "The Flash", which tells the story of the last man in the world where entire humanity has returned into mindless human killers. New and crowded community afraid of him and ones like him and is trying to destroy them. The game starts exactly at the point where the main character realizes that the only way to protect from them is to kill all of them, but there is no enemy in this game. The game play is time-action-based with no strategic or role-playing elements; instead, the game entirely provides a rush of adrenaline. The only goal is to collect coins in the allotted time by crossing hurdles. The version in this project consists of levels.

1.5.1 Start Window of the Game

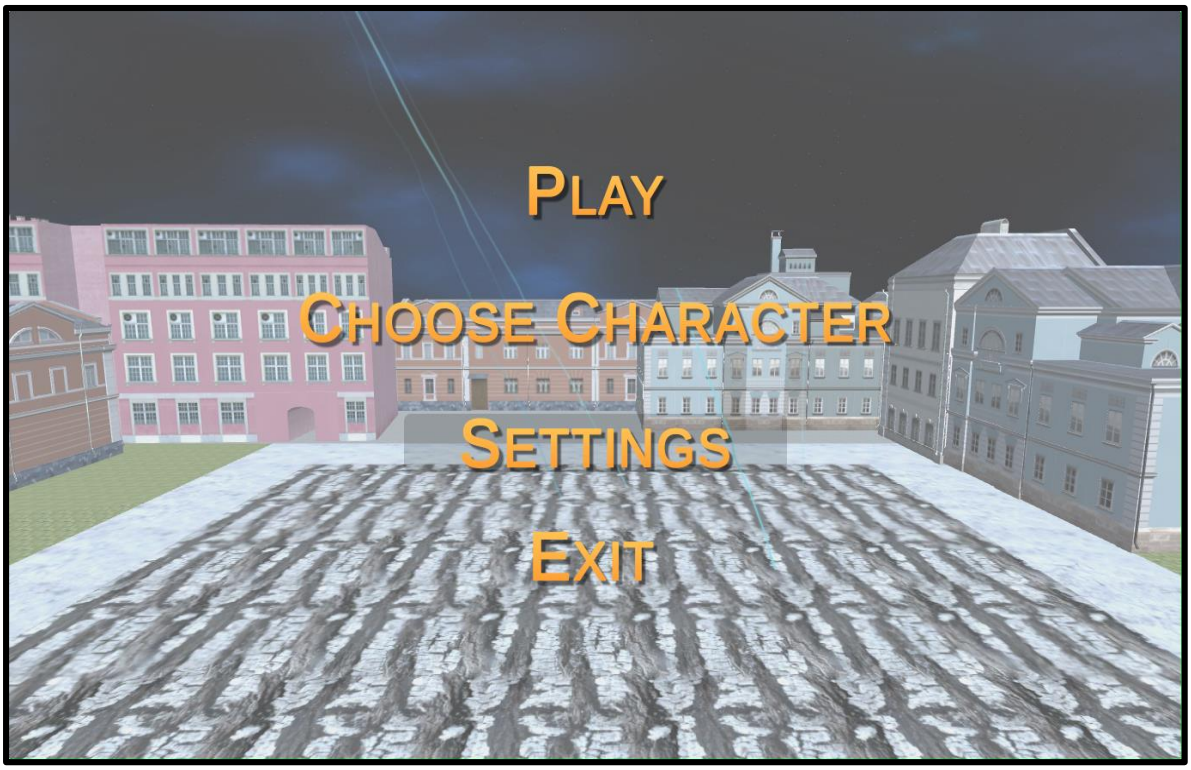


Fig 1.3 Start Window of Game

The player experiences the game world through the eyes of a main character. This makes one feel one is a part of the game. The player controls the main character, who is one of the healthy survivors. The abilities of our character as a player are walking and running in all four directions. The player starts the game with a weapon of lightening. The effect of damage is collision with hurdles. The start window of game as shown in Fig 1.3.

1.5.2 Choose Characters

You have to choose character by clicking on button “CHOOSE CHARACTER”. There is a male character and a female character. The below window appears for selecting the character as shown in fig 1.3.

Fig 1.4 (a) Choose Character Window (Female)

Fig 1.4 (b) Choose Character Window (Male)

1.5.3 Environment Type of the Game

You can play this 3-D game in two modes:

- LIGHT Mode
- DARK Mode

These modes you can see previously in fig 1.1(a) and fig 1.2. You can select your comfortable mode from Settings panel as shown in fig 1.5.

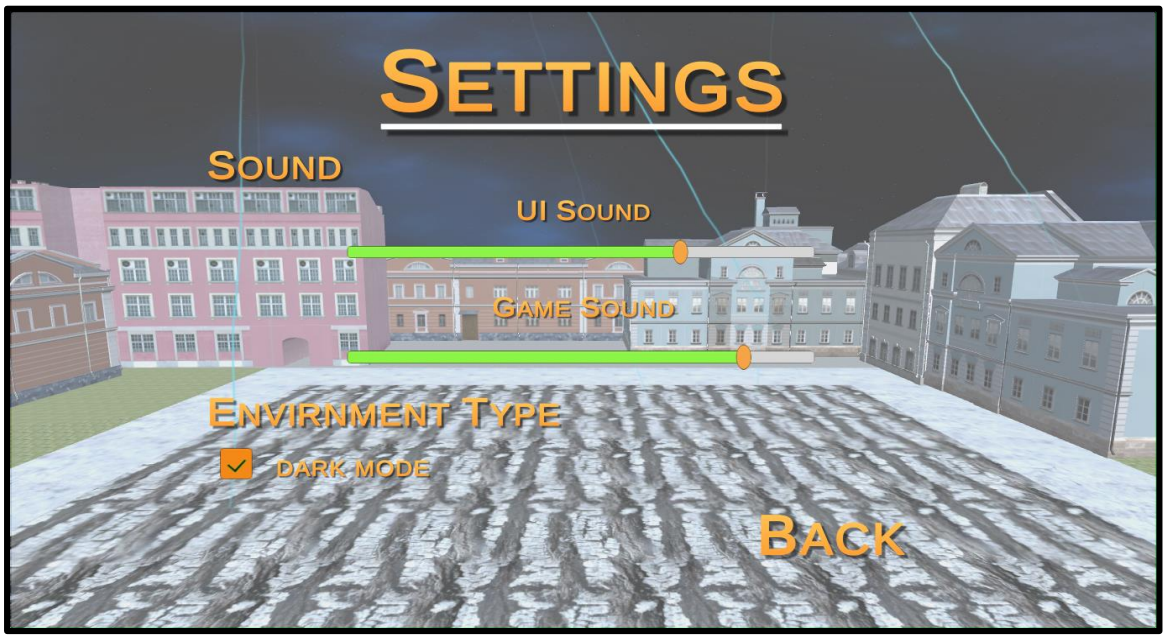


Fig 1.5 Environment Type Window

1.5.4 Game Sound and UI Sound

You can adjust Game sound and UI sound accordingly by these bars as shown in fig 1.5 respectively. You can increase and decrease the sound by sliding forward and backward green bars respectively.

1.5.5 Play the Game

You can play the game by clicking on PLAY button which is available on Start window. The character which you have chosen starts running on the road and execute given task. This is shown in fig 1.6. Coins collected, Current Score and Level has been also shown to the player while playing.

Fig 1.6 Game Playing Window

1.5.6 Resume/Exit/Back Main Menu

- To Resume the Game: Click on RESUME button.
- To Continue the Game: Click on START AGAIN button.
- To go back to Main Menu: Click on BACK MAIN MENU button.
- To Exit the Game: Click on EXIT button.

You can easily choose the button by clicking on it from this window as shown in fig 1.7.

Fig 1.7 Resume/Exit/Back Main Menu panel

1.5.7 Score Card of Current Level in Game

The score of the player will be shown when the player collide with a hurdle. There will be two scores shown to the player as shown in Fig 1.8:

- Current Score
- High Score

Fig 1.8 Score Panel

At high score, coin collected in the distance travelled is shown. The total is the sum of high score and coins collected.

At current score, coin collected in the distance travelled is shown. The total is the sum of current score and coins collected.

1.6 ENVIRONMENT OF THE GAME

Environment of the game includes:

- Buildings
- Tress
- Benches
- Roads
- Grass
- Cars and Trucks

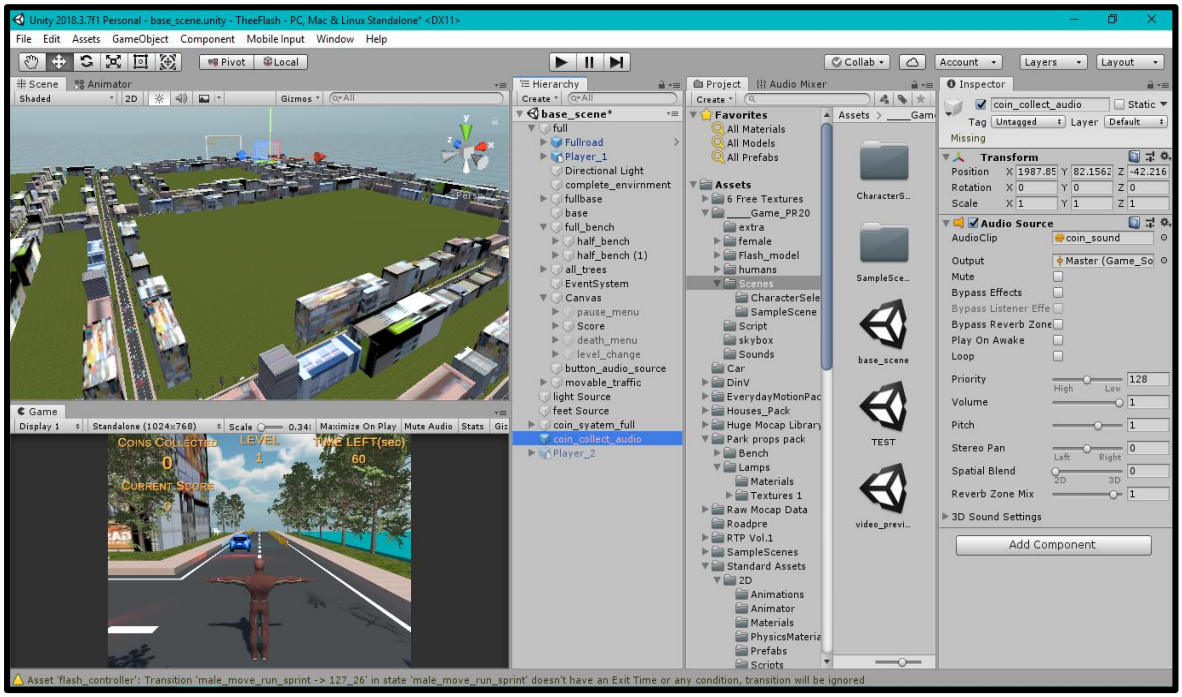


Fig 1.9 Environment of the Game

CHAPTER 2

INTRODUCTION TO 3Ds MAX

2.1 SOFTWARE OVERVIEW

3Ds Max is a powerful 3D modeling, animation, effects, and rendering solution that has been used in everything from video games to feature films. 3Ds Max has a wealth of features that can tackle almost any sort of project and generate incredibly realistic or highly stylized images. 3Ds Max's large feature set may seem daunting to the newcomer, but the software has a consistent and easy-to-use interface, and this chapter introduces that interface. Autodesk 3ds Max, formerly 3D Studio Max, is a professional 3D computer graphics program for making 3D animations, models, games and images. It is developed and produced by Autodesk Media and Entertainment. It has modeling capabilities, a flexible plugin architecture and can be used on the Microsoft Windows platform. It is frequently used by video game developers, many TV commercial studios and architectural visualization studios. It is also used for movie effects and movie pre-visualization. In addition to its modeling and animation tools, the latest version of 3ds Max also features shades, dynamic simulation, particle systems, radiosity, normal map creation and rendering, global illumination, a customizable user interface, and its own scripting language.

2.2 FEATURES OF 3D - MAX

- **MAXScript:** MAXScript is a built-in scripting language that can be used to automate repetitive tasks, develop new tools and user interfaces, and much more.
- **Character Studio:** Character Studio was a plugin which since version 4 of Max is now integrated in 3D Studio Max, helping users to animate virtual characters.
- **Scene Explorer:** Scene Explorer, a tool that provides a hierarchical view of scene data and analysis, facilitates working with more complex scenes. Scene Explorer has

the ability to sort, filter, and search a scene by any object type or property (including metadata).

- **DWG import:** 3Ds Max supports both import and linking of DWG files.
- **Texture assignment/editing:** 3Ds Max offers operations for creative texture and planar mapping, including tiling, mirroring, decals, angle, rotate, blur, UV stretching, and relaxation; Remove Distortion; Preserve UV; and UV template image export.
- **General Keyframing**: Two keying modes: set key and auto key, offer support for different keyframing workflows. Fast and intuitive controls for keyframing including cut, copy, and paste let the user create animations with ease.
- **Constrained animation:** Objects can be animated along curves with controls for alignment, velocity, smoothness, and looping, and along surfaces with controls for alignment.
- **Skinning:** Either the Skin or Physique modifier may be used to achieve precise control of skeletal deformation, so the character deforms smoothly as joints are moved, even in the most challenging areas, such as shoulders.
- **Skeletons and inverse kinematics (IK):** Characters can be rigged with custom skeletons using 3ds Max bones and rigging tools powered by Motion Capture Data. All animation tools including expressions, scripts, list controllers, and wiring can be used along with a set of utilities specific to bones to build rigs of any structure and with custom controls, so animators see only the UI necessary to get their characters animated.

2.3 ADOPTION

Many films have made use of 3Ds Max, or previous versions of the program under previous names, in CGI animation, such as Avatar and 2012, which contain computer generated graphics from 3Ds Max alongside live-action acting. Mudbox was used in the final texturing of the set and characters in Avatar, with 3ds Max and Mudbox being closely related. 3Ds Max has been used in the development of 3D computer graphics for a number of video games. Architectural and engineering design firms use 3Ds Max for developing concept art and previsualization. 3Ds Max shares a close resemblance with AutoCAD. Educational close program at secondary and tertiary level use 3ds Max in their courses on

3D computer graphics and computer animation. Students in the FIRST competition for 3d animation are known to use 3ds Max.

2.4 MODELING TECHNIQUES

- **Polygon Modeling:** Polygon modeling is more common with game design than any other modeling technique as the very specific control over individual polygons allows for extreme optimization. Versions 4 and up feature the Editable Polygon object, which simplifies most mesh editing operations, and provides subdivision smoothing at customizable level. Version 7 introduced the edit poly modifier, which allows the use of the tools available in the editable polygon object to be used higher in the modifier stack as shown in Fig 2.1.

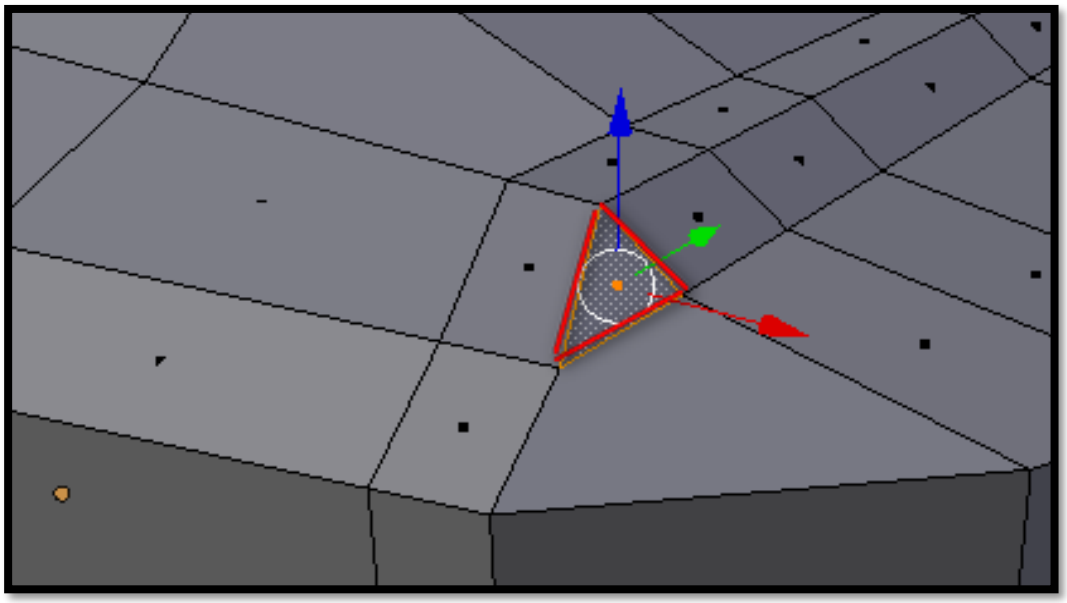


Fig 2.1 Polygon Modeling

- **NURBS or non-uniform rational B-spline**: An alternative to polygons, it gives a smoothed out surface that eliminates the straight edges of a polygon model. NURBS is a mathematically exact representation of freeform surfaces like those used for car bodies and ship hulls, which can be exactly reproduced at any resolution whenever needed. With NURBS, a smooth sphere can be created with only face as shown in Fig 2.2.

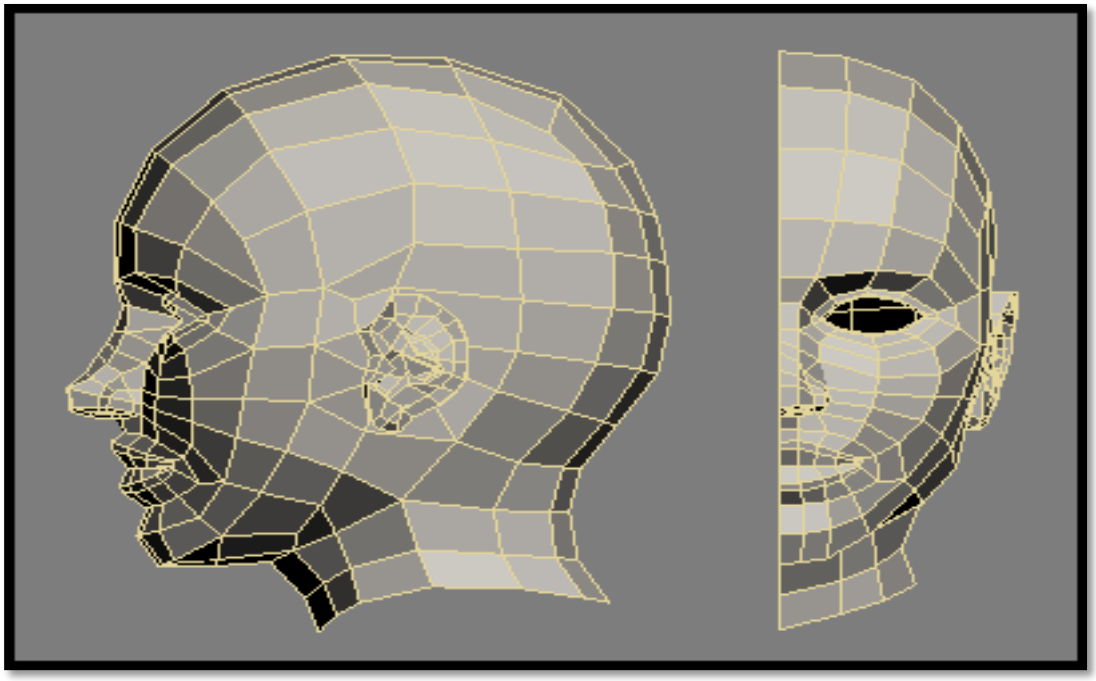


Fig 2.2 NURBS rational B-spline

- **Surface tool/Editable patch object:** This modifier makes a surface from every 3 or 4 vertices in a grid. This is often seen as an alternative to “mesh” or “nurbs” modelling, as it enables a user to interpolate curved sections with straight geometry (like a hole through a box shape, human’s legs, hands) as shown in Fig 2.3.

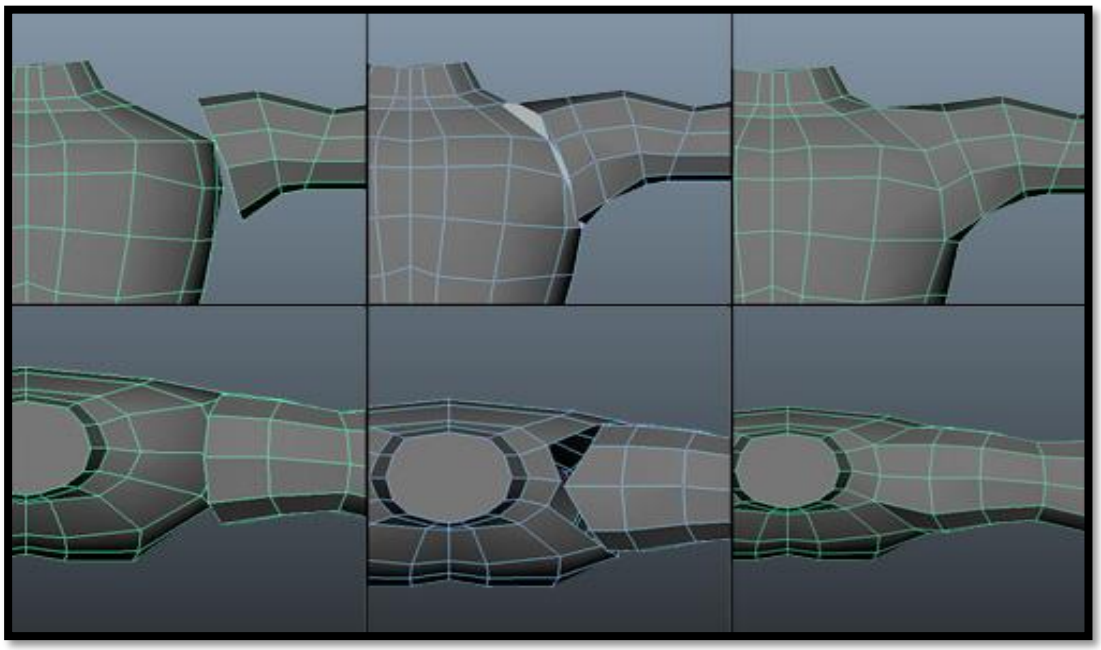


Fig 2.3 Surface Modeling

2.5 PREDEFINED PRIMITIVES

This is a basic method, in which one models using only boxes, spheres, cones, cylinders and other predefined objects from the list of Predefined Standard Primitives a list of Predefined Extended Primitives. One may also apply Boolean operations, including subtract, cut and connect. For example, one can make two spheres which will work as blobs that will connect with each other. These are called meta-balls.

2.6 UNDERSTANDING THE 3D-MAX INTERFACE

Understanding the 3d-Max Interface is the foundation of everything you'll do within the software, including modeling objects, creating textures, animating, and final rendering. The program that be covered in this manual Autodesk 3ds Max. At first, 3ds Max may seem a bit over-whelming its four screens and multiple command bars and menus, but in time all those characteristics will become almost natural. When 3ds max opens, the screen with an object like cube, cuboid, sphere, etc. as shown below in Fig 2.4.

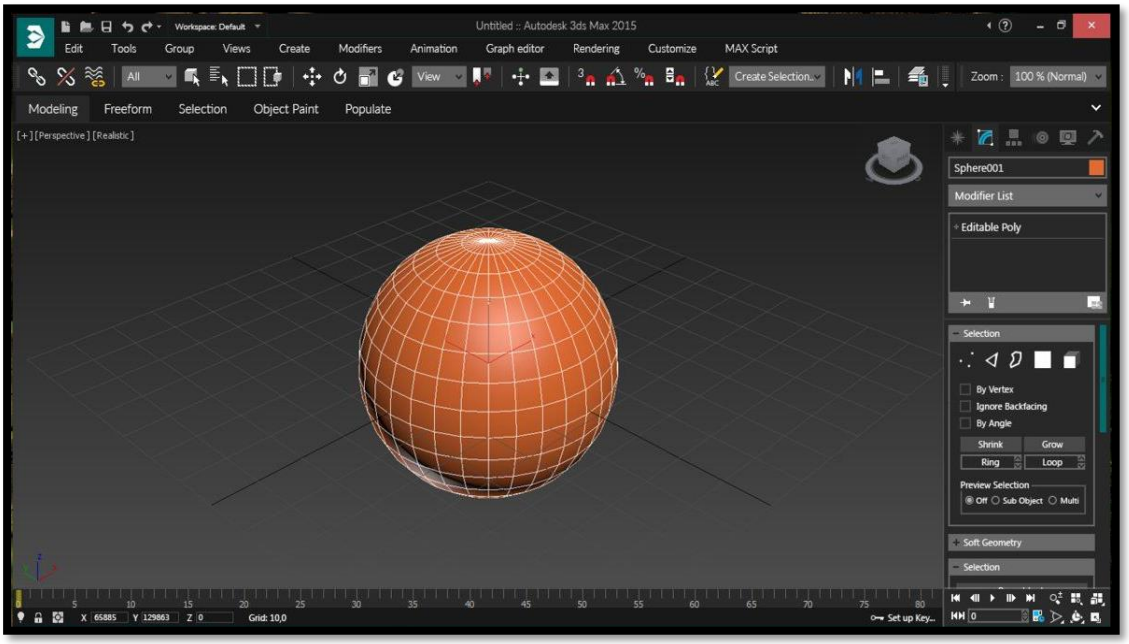


Fig 2.4 GUI of 3D-Max

The four mini screens in Fig 2.1 are known as the viewports and are useful for placement of and modeling objects. The cube in the upper right comer of every viewport is

called the view cube. It allows to rotate the entire scene. In the upper left comer of every viewport are three little menus. The plus sign allows one to maximize a viewport, which basically means to make it the sole viewport (not recommended unless the four viewports just really annoy one), activate or disable a view port, show or hide the view cube (definitely not recommended in any situation. It may be evil sometimes but it makes things so much harder without it) and so on. The menu next to the plus sign allows one to change the orientation of the viewport to perspective, left, right, front, back, top, bottom, or orthographic. It also allows one to enter into the view from a camera, select lights, etc. The final menu allows one to change how the objects are viewed in the viewports.

2.6.1 User Interface Elements

Fig 2.5 shows the Max Interface elements. At the very top left of the application window is a large button called Application, clicking it opens the Application menu, which provides access to many file operations. Also running along the top is the Quick Access toolbar, which provides access to common commands, and the InfoCenter, which offers support for various Autodesk applications. Some of the most important commands in the Quick Access toolbar are file management commands such as Save File and Open File. If you do something and then wish you hadn't, you can click the Undo Scene Operation icon or press Ctrl+Z. To redo a command or action that you just undid, click the Redo Scene Operation button or press Ctrl+Y.

The Interface elements are:

- **Application button:** Opens Application menu that provides file management commands.
- **Main toolbar:** Provides quick access to tools and dialog boxes for many of the most common tasks.
- **Graphite Modeling Tools ribbon:** Provides access to a wide range of tools to make building and editing models in 3ds Max fast and easy.
- **Quick Access toolbar:** Provides some of the most commonly used file management commands, as well as Undo and Redo.

- **Menu bar:** Provides access to commands grouped by category.
- **InfoCenter:** Provides access to information about 3ds Max and other Autodesk products.
- **Command panel tabs:** Where all the editing of parameters occurs; provides access to many functions and creation of options; divided into tabs that access different panels, such as Creation pane, Modify panel, etc.
- **Rollout:** A section of the Command panel that can expand to show a listing of parameters or collapse to just it's heading name.
- **Viewports:** You can choose different views to display in these four viewports as well as different layouts from the viewport label menus.
- **Time slider:** Shows the current frame and allows for changing the current frame by moving (or scrubbing) the time bar.
- **Track bar**: Provides a timeline showing the frame numbers; select an object to view its animation keys on the track bar.
- **Prompt line and status bar controls:** Prompt and status information about your scene and the active command.
- **Coordinate display area:** Allows you to enter transformation values.
- **Animation keying control:** Animation playback controls.
- **Viewport navigation controls:** Icons that control the display and navigation of the viewports; icons may change depending on the active viewport.

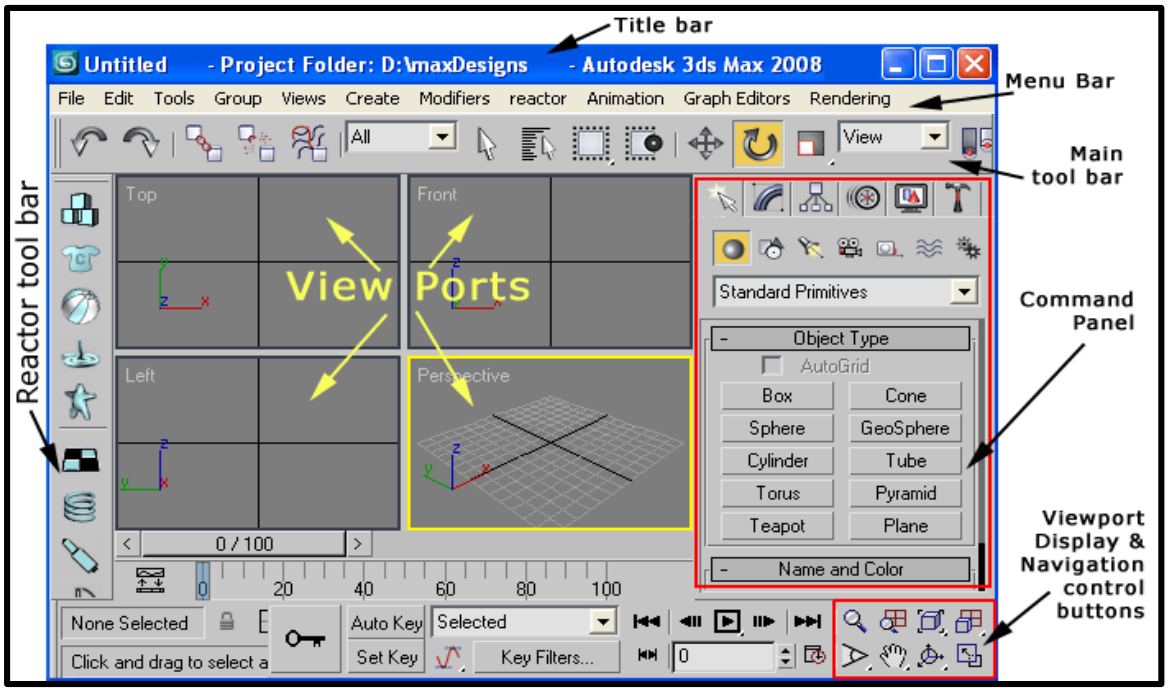


Fig 2.5 User Interface Elements of 3-D max

Just below the Quick Access toolbar is the menu bar, which runs across the top of the UI. The menus give you access to a ton of commands—from basic scene operations, such as Undo under the Edit menu, to advanced tools such as those found under the Modifiers menu. Immediately below the menu bar is the Main toolbar. It contains several icons for functions such as the three Transform tools: Move, Rotate, and Scale.

2.6.2 Mouse Buttons

Each of the three buttons on the mouse, in Fig 2.6, plays a slightly different role when manipulating viewports in the workspace. When used with modifiers such as the Alt key, they are used to navigate the scene.

Fig 2.6 Mouse Buttons

2.6.3 Quad Menus

When you click the right mouse button anywhere in an active viewport, except on the viewport label, a quad menu is displayed at the location of the mouse cursor. The quad menu in Fig 2.7, can display up to four quadrant areas with various commands without your having to travel back and forth between the viewport and rollouts on the Command panel. The two right quadrants of the default quad menu display generic commands, which are shared between all objects. The two left quadrants contain context-specific commands, such as mesh tools and light commands. You can also repeat your last quad menu command by clicking the title of the quadrant.

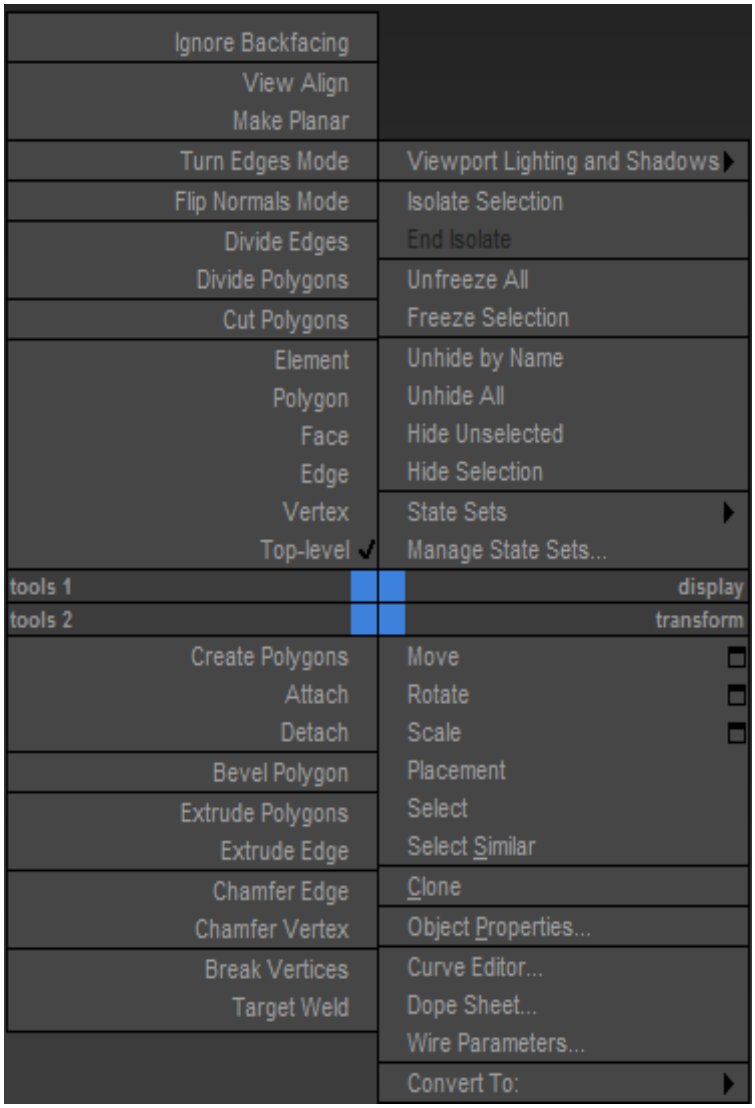


Fig 2.7 Quad Menu

2.6.4 Graphite Modeling Tools Ribbon

The Graphite Modeling Tools (also called the Graphite Modeling Tools ribbon), is a section of the UI directly under the mam toolbar. The Graphite Modeling Tools ribbon provides you with a wide range of tools to make building and editing models in 3ds Max fast and easy. All the available tools are divided into tabs that are organized by function, and then further divided into panels. For example, the Graphite Modeling Tools tab in Fig 2.8, contains the tools you use most often for polygon modeling and editing, organized into separate panels for easy, convenient.

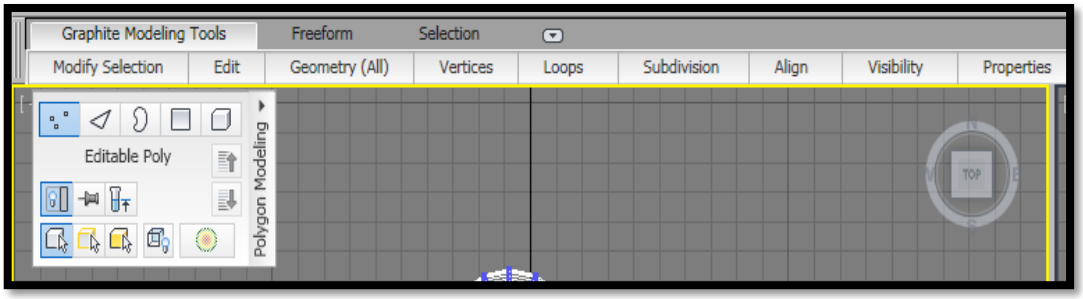


Fig 2.8 Graphite Modeling Tool Ribbon

The panels found in the Polygon Modeling tab are:

- Polygon modeling panel
- Modify Selection panel
- Edit panel
- Geometry (All) panel
- Subdivision panel
- Loops panel
- Align panel
- Visibility panel
- Properties

2.6.5 Command Panel

Everything you need to create, manipulate, and animate objects can be found in the Command panel running vertically on the right side of the UI. The Command panel is divided into tabs according to function as shown in Fig 2.9. The function or toolset you need to access all determine tab you need to click. When you encounter a panel that is longer than your screen, 3ds Max displays a thin vertical scroll bar on the right side. Your cursor also tums into a hand that lets you click and drag the panel up and down.

The functions of command panel are as follows:-

- **Create panel:** Lets you create objects, lights, cameras, etc.
- **Modify panel:** Lets you apply and edit modifiers to objects.
- **Hierarchy panel:** Lets you adjust the hierarchy for objects and adjust their pivot points.

- **Motion panel:** Lets you access animation tools and functions.
- **Display panel:** Lets you access display options for scene objects.
- **Utilities panel:** Lets you access several functions of 3ds Max, such as motion capture utilities and the Asset Browser.

The Command panel and all its tabs give you access to an object's parameters.

Parameters are the values that define a specific attribute of or for that object. For example, when an object is selected in a viewport, its parameters are shown in the Modify panel, where you can adjust them. When you create an object, that object's creation parameters are shown (and editable) in the Create panel. In the Modify panel you'll find the modifier stack. This UI element lists all the modifiers that are active on any selected object. Modifiers are actions applied to an object that change it somehow, such as bending or warping. You can stack modifiers on top of each other when creating an object and then go back and edit any of the modifiers in the stack (for the most part) to adjust the object at any point in its creation.

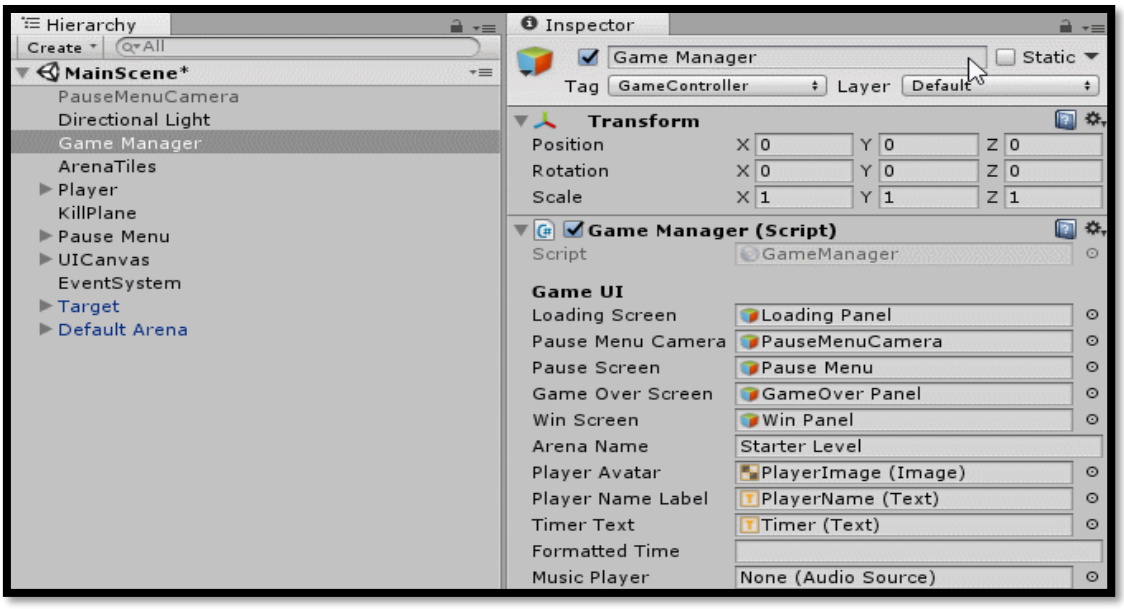


Fig 2.9 Control Panel

An object or mesh in 3ds Max composed of polygons that define the surface For example, the facets or small rectangles on a sphere are polygons. All connected at common edges at the correct angles and in the proper arrangement to make a sphere. The point that generate a polygon are called vertices. The lines that connect the points are called edges.

Polygons, vertices, and edges are examples of sub-objects and arc all editable go that you can fashion any of surface or mesh shape.

2.6.6 Time Slider and Track Bar

Running across the bottom of the 3Ds Max UI are the time slider and the track bar. The time slider allows you to move through any frame in your scene by scrubbing (moving the slider back and forth). You can move through your animation one frame at a time by clicking on the arrows on either side of the time slider or by pressing the < and > keys. The track bar is directly below the time slider. The track bar is the timeline that displays the timeline format for your scene. More often than not, the track bar is displayed in frames, with the gap between each tick mark representing frames. On the track bar, you can move and edit your animation properties for the selected object. When a key frame is present, right-click it to open a context menu where you can delete key frames, edit individual transform values, and filter the track bar display.

2.6.7 File Management

3Ds Max provides several subfo1ders automatically grouped into projects for you. Different kinds of files are saved categorized folders under the project folder. For example, scene files are saved in a Scenes folder and rendered images are saved in a Render Output folder within the project folder [2].

CHAPTER 3

INTRODUCTION TO UNITY

3.1 OVERVIEW

Unity is made to empower you to create the best interactive entertainment or multimedia experience that you can. Unity is a powerful engine with a variety of tools that can be utilized to meet your specific needs. The editor is intuitive and customizable allowing you a greater freedom in your workflow. This section is the key to getting started with Unity. It will follow through the important steps for creating a game in Unity; starting with the asset workflows, then how to build up your scenes and finally to publishing your build.

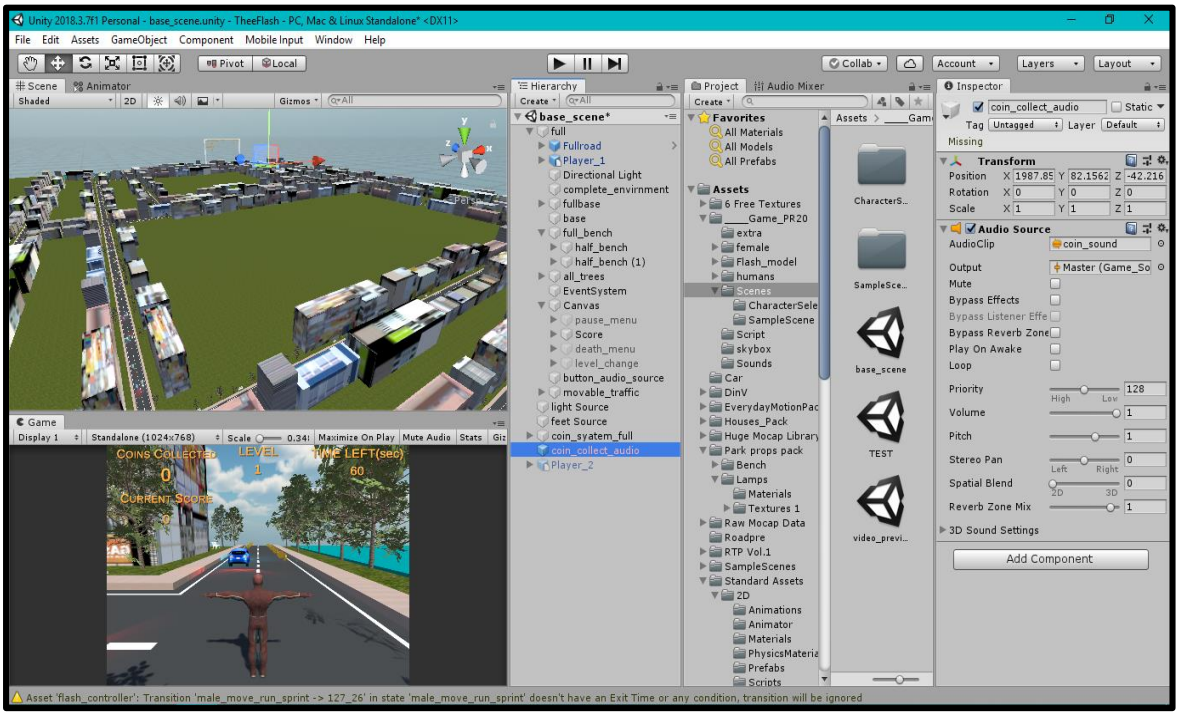


Fig 3.1 Unity Build up Scenes

Unity was launched at Apple's Worldwide Developers Conference to build functions and projects on Mac computers. It was later was introduced for other platforms. Unity performs

as a game development environment providing intuitive tools that help you design awesome 3D content. It also provides cross platform publishing, millions of ready-made assets in the Asset Store and an online community. For individual developers and studios, Unity's environment reduces the time and cost allowing you to develop uniquely beautiful interactive games. It provides flexibility to deploy projects on multi-platforms like iOS, Windows and android.

Unity has three parts:

- A game engine: This allows the games to be created, animate and played in different environment.
- An application where the design or the interface is put together with a graphics previous option and control play functions.
- A code editor: The IDE provides a text editor to write code. However, a separate text editor is often used to avoid confusion.

3.2 UNITY BASICS

This section will explain the Unity interface turns, using objects, creating environments. Coordinate Space: Three axes X, Y and Z –based on the Cartesian coordinate system.

3.2.1 Learning the Interface

- File > New Project

You don’t have to import any of the packages at this point, but do specify the save to location in the dialog window. Remember to keep all you project related assets in this location to avoid missing files and broken links later in the game development process as shown in Fig 3.2(a) and Fig 3.2(b).

- File > Save Scene

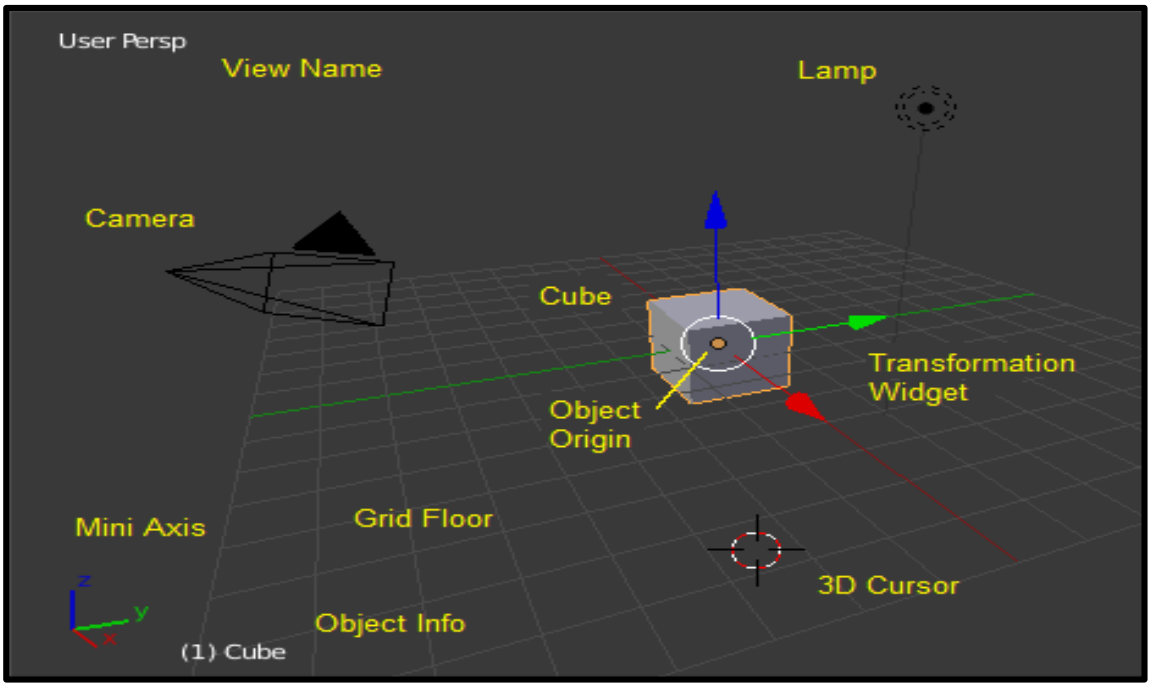


Fig 3.2 (a) Components exiting in Unity Interface

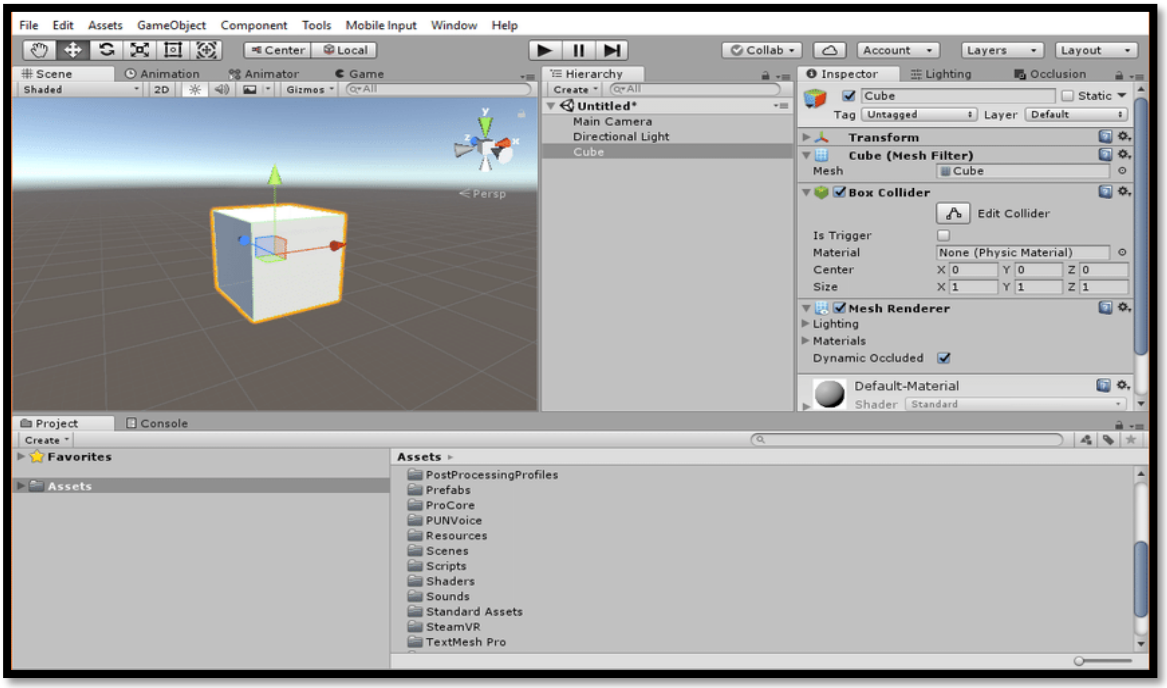


Fig 3.2 (b) Unity Interface

3.2.2 Project Browser

In this view, you can access and manage the assets that belong to your project as shown in fig 3.3

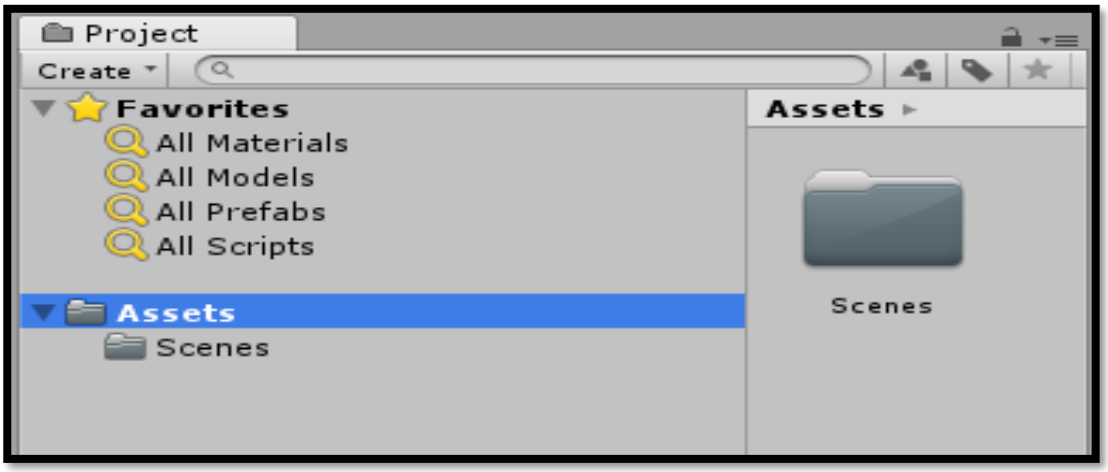


Fig 3.3 Project Browser

View the left panel of the browser shows the folder structure of the project as a hierarchical list as in fig 3.2 and 3.3. When a folder is selected from the list by clicking, its contents will be shown in the panel to the right. The individual assets are shown as icons that indicate their type (script, material, sub-folder, etc.). The icons can be resized using the slider at the bottom of the panel; they will be replaced by a hierarchical list if the slider is moved to the extreme left. The space to the left of the slider shows the currently selected item, including a full path to the item if a search is being performed. Above the project structure list is a Favorites section where you can keep frequently-used items for easy access. You can drag items from the project structure list to the Favorites and also save search queries there.

Just above the panel is a "breadcrumb trail" that shows the path to the folder currently being viewed. The separate elements of the trail can be clicked for easy navigation around the folder hierarchy. When searching, this bar changes to show the area being searched (the root Assets folder, the selected folder or the Asset Store) along with a count of free and paid assets available in the store, separated by a slash. There is an option in the General section of Unity's Preferences window to disable the display of Asset Store hit counts if they are not required. Located at the left side.

3.2.3 Searching Options in Unity

There are various searching in unity as you can see in Fig 3.4. These are:-

- Search by Label
- Search by Type
- Save Search
- Search Box
- Saved Search

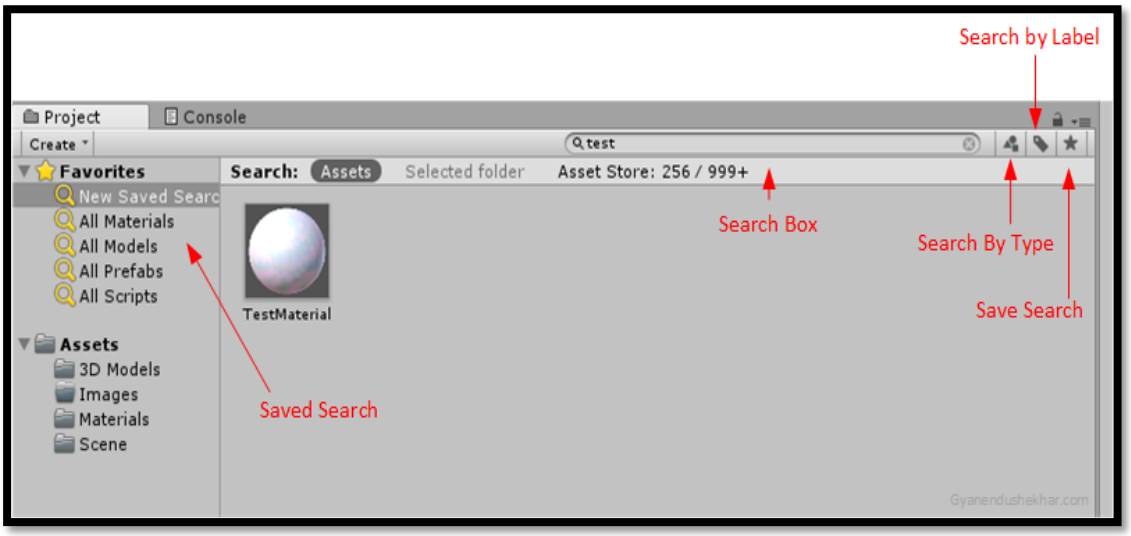


Fig 3.4 Search Options

3.2.4 Searching the Asset Store

The Project Browser's search can also be applied to assets available from the Unity Asset Store. If you choose Asset Store from the menu in the breadcrumb bar, all free and paid items from the store that match your query will be displayed.

Searching by type and label works the same as for a Unity project The search query words will be checked against the asset name first and then the package name, package label and package description in that order (so an item whose name contains the search terms Will be ranked higher than one With the same terms in its package description).

3.2.5 Shortcut Keys

Table 3.1 Short-cuts of Unity

COMMAND	USE
F	Frame selected (i.e., show the selected asset In Its containing folder)
Tab	Shift focus between first column and second column(Two columns)
Ctrl/Cmd + F	Focus search field
Ctrl/Cmd + A	Select all visible items in list
Ctrl/Cmd + D	Duplicate selected assets
Delete	Delete with dialog
Delete + Shift	Delete without dialog
Backspace + Cmd	Delete without dialog
Enter	Begin rename selected
Cmd + Down Arrow	Open selected assets
Cmd + Up Arrow	Jump to Parent folder
F2	Begin rename selected
Backspace	Jump to Parent Folder(Win, Two Columns)
Right Arrow	Expand selected item (Tree Views And Search Results)
Left Arrow	Collapse selected item (tree views and search results)
Alt + Right Arrow	Expand item when showing assets as previews
Alt + Left Arrow	Collapse item when showing assets as previews

3.3 HIERARCHY AND PARENTING

The Hierarchy contains every Game Object in the current Scene. Some of these are direct instances of asset files like 3D models, and others are instances of Prefabs, custom objects that will make up much of your game. You can select objects in the Hierarchy and drag one object onto another to make use of Parenting. As objects are added and removed in the scene, they will appear and disappear from the Hierarchy as well Unity uses a concept called Parenting. To make any Game Object the child of another, drag the desired child onto the desired parent in the Hierarchy.

A child will inherit the movement and rotation of its parent. You can use a parent object's foldout arrow to show or hide its children as necessary Fig 3.4 shows the hierarchy of an object.

3.4 ASSET WORKFLOW

Here we'll explain the steps to use a single asset with Unity. These steps are general and are meant only as an overview for basic actions. For the example, we'll talk about using a 3D mesh.

3.4.1 Create Rough Asset and Import

Use any supported 3D modeling package to create a rough version of your asset. When you save your asset initially, you should save it normally to the Assets folder in your Project folder. When you open the Unity project, the asset will be detected and imported into the project. When you look in the Project View, you'll see the asset located there, right where you saved it. Please note that Unity uses the FBX exporter provided by your modeling package to convert your models to the FBX file format. You will need to have the FBX exporter of your modeling package available for Unity to use. Alternatively, you can directly export as FBX from your application and save in the Projects folder.

3.4.2 Import Settings and Adding Assets to the Scene

If you select the asset in the Project View the import settings for this asset will appear in the Inspector. The options that are displayed will change based on the type of asset that is selected. Simply click and drag the mesh from the Project View to the Hierarchy or Scene View to add it to the Scene. When you drag a mesh to the scene, you are creating a Game Object that has a Mesh Renderer Component. If you are working with a texture or a sound file, you will have to add it to a Game Object that already exists in the Scene or Project.

3.4.3 Putting Different Assets Together

Here is a brief description of the relationships between the most common assets: A Material is applied to a GameObject (with a Mesh Renderer Component), an Animation is applied to a GameObject (with an Animation Component), a sound file is applied to a GameObject (with an Audio Source Component) and a texture is applied to the material.

3.4.4 Creating a Prefab

A Prefab is a stored collection containing one or more complete Game Objects with components attached and properties set. Prefabs are asset types, so they do not appear in the scene in their own right. However, they can be used to create instances of the stored objects in the scene. Each instance is a copy of the original prefab. For example, you might use a prefab to store a tree object and then create many instances of the tree in a forest scene. By default, changes made to the prefab are automatically applied to all the instances, and so using prefabs can be a good way to maintain consistency among a set of objects. However, you can also the link between the instance and the prefab if you want to create special variations on the original. You can also make changes to an instance, and then save those changes to the prefab (menu: GameObject > Apply Changes to Prefab).

3.4.5 Updating Assets

You have imported, instantiated, and linked your asset to a Prefab. Now when you want to edit your source asset, just double-click it from the Project View. The appropriate application will launch, and you can make any changes you want. When you're done updating it just save it. Then when you switch back to Unity, the update will be detected, and the asset will be re-imported. The asset's link to the prefab will also be maintained. So the effect you will see is that your Prefab will update. That’s all you have to know to update assets. Just open it and save.

3.5 CREATING SCENES

Scenes contain the objects of your game. They can be used to create a main menu, individual levels, and anything else. Think of each unique Scene file as a unique level. In each Scene, you will place your environments, obstacles, and decorations, essentially designing and building your game in pieces.

3.5.1 Instancing Prefabs and Adding Components and Scripts

Once you've created a Prefab, you can quickly and easily make copies of the Prefab, called an Instance. To create an instance of any Prefab, drag the Prefab from the Project View to the Hierarchy or scene View. Now you have a unique instance of your Prefab to position and tweak as you like. When you have a Prefab or any, GameObject highlighted, you can add additional functionality to it by using Components. Scripts are a type of Component. To add a Component, just highlight your GameObject and select a Component from the Component menu. You will then see the Component appear in the Inspector of the GameObject. Scripts are also contained in the Component menu by default. If adding a Component breaks the Game Object's connection to its Prefab, you can always use GameObject->Apply Changes to Prefab from the menu to re-establish the link.

3.5.2 Placing Game Objects and Working with Cameras

Once your GameObject is in the scene, you can use the Transform Tools to position it wherever you like. Additionally, you can use the Transform values in the Inspector to fine-tune placement and rotation. Cameras are the eyes of your game. Everything the player will see while playing is through one or more cameras. You can position, rotate, and parent cameras just like any other Game Object. A camera is just a GameObject with a Camera Component attached to it. Therefore it can do anything a regular GameObject can do, and then some camera-specific functions too. There are also some helpful Camera scripts that are installed with the Scripts package. The Scripts package can be included when you create a new project, or you can use the Assets->Import Package... menu. The scripts that

you import can be found in Components->Camera-Control from the menu. There are some additional aspects to cameras which will be good to understand.

3.5.3 Lights

Except for some very few cases, you will always need to add Lights to your scene. There are three different types of lights, and all of them behave a little differently from each other. The important thing is that they add atmosphere and ambience to game. Different lighting can completely change the mood of your game, and using lights effectively will be an important subject to learn.

3.5.4 Unity Hotkeys

The basic keys for various tools are given in the table below:

Table 3.2 Keys for Tools

KEYSTROKE	COMMAND
Q	Pan
W	Move
E	Rotate
R	Scale
T	Rect Tool
Z	Pivot Mode Toggle
X	Pivot Rotation Toggle
V	Vertex Snap
CTRL/CMD+LMB	Snap

The basic keys for the GameObject are given in the table below:

Table 3.3 Keys for GameObject

KEYSTROKE	COMMAND
CTRL/CMD+ALT+F	Move to view
CTRL/CMD+SHIFT+F	Align with view
SHIFT+F or double-F	Locks the scene view camera to the selected GameObject
CTRL/CMD+SHIFT+N	New game object

The basic keys for the Window are given in the table below:

Table 3.4 Keys for Window

KEYSTROKE	COMMAND
CTRL/CMD+2	Game
CTRL/CMD+3	Inspector
CTRL/CMD+4	Hierarchy
CTRL/CMD+5	Project
CTRL/CMD+6	Animation
CTRL/CMD+7	Profiler
CTRL/CMD+9	Asset store
CTRL/CMD+0	Animation
CTRL/CMD+SHIFT+C	Console
CTRL/CMD+1	Scene

The basic keys for the Edit options are given in the table below:

Table 3.5 Keys for Edit Tools

KEYSTROKE	COMMAND
CTRL+Y (Windows only)	Redo
CMD+SHIFT+Z (Mac only)	Redo
CTRL/CMD+X	Cut
CTRL/CMD+C	Copy
CTRL/CMD+V	Paste
CTRL/CMD+D	Duplicate
SHIFT+Del	Delete
F	Frame(center) selection
CTRL/CMD+F	Find
CTRL/CMD+A	Select All
CTRL/CMD+P	Play
CTRL/CMD+SHIFT+P	Pause
CTRL/CMD+ALT+P	Step
CTRL/CMD+Z	Undo

The basic keys for the Asset is given in the table below:

Table 3.6 Key for Asset

KEYSTROKE	COMMAND
CTRL/CMD+R	Refresh

The basic keys for the Selection process are given in the table below:

Table 3.7 Keys for Selection Procedure

KEYSTROKE	COMMAND
CTRL/CMD+SHIFT+2	Load Selection 2
CTRL/CMD+SHIFT+3	Load Selection 3
CTRL/CMD+SHIFT+4	Load Selection 4
CTRL/CMD+SHIFT+5	Load Selection 5
CTRL/CMD+SHIFT+6	Load Selection 6
CTRL/CMD+SHIFT+7	Load Selection 7
CTRL/CMD+SHIFT+8	Load Selection 8
CTRL/CMD+SHIFT+9	Load Selection 9
CTRL/CMD+ALT+I	Save
CTRL/CMD+SHIFT+2	Save Selection 2
CTRL/CMD+SHIFT+3	Save Selection 3
CTRL/CMD+SHIFT+4	Save Selection 4
CTRL/CMD+SHIFT+5	Save Selection 5
CTRL/CMD+SHIFT+6	Save Selection 6
CTRL/CMD+SHIFT+7	Save Selection 7
CTRL/CMD+SHIFT+8	Save Selection 8
CTRL/CMD+SHIFT+9	Save Selection 9
CTRL/CMD+ALT+I	Load Selection 1

3.6 IMPORTANT FEATURES OF UNITY

- **Rendering:** It provides features to render text and use of shadow maps for dynamic shadows. Various file formats of different software are supported. For instance, Adobe Photoshop, Blender and 3ds Max are supported.

- **Scripting:** Scripting is built on Mono, the open source platform for .NET Framework. Programmers write the Unity Script similar to JavaScript, C sharp and Boo.
- **Asset Tracking:** Unity has an asset server- control solution for developer scripts and game assets. It also has a terrain and vegetation engine built in global illumination and light mapping and built-in path finding meshes.
- **Physics:** The Unity engine provides built-in support for PhysX physics engine with real time cloth simulation on skinned meshes, collision layers and thick ray casts.

3.8 ADVANTAGES OF UNITY

The advantages of Unity are mentioned below:

- Platform Support
- IDE
- Great Graphics
- Documentation
- Code is very much stable in comparison to other languages and consists of a great architecture for good performance and reduced errors.
- It comes with an easy profiler that is used for game optimization and prevents memory leaks.
- The biggest benefit is productivity. I have seen things come together at high quality in a short period of time using Unity.
- When compared against building your own engine, it’s much less effort. You can be up in running with a prototype in hours and not take months to first build an engine [1].

CHAPTER 4

INTRODUCTION TO BLENDER

4.1 INTRODUCTION

Blender Foundation is a Dutch public-benefit corporation, established to support and facilitate the projects on blender.org.

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Advanced users employ Blender's API for Python scripting to customize the application and write specialized tools; often these are included in Blender's future releases. Blender is well suited to individuals and small studios who benefit from its unified pipeline and responsive development process. Examples from many Blender-based projects are available in the showcase.

Blender is cross-platform and runs equally well on Linux, Windows, and Macintosh computers. Its interface uses OpenGL to provide a consistent experience. To confirm specific compatibility, the list of supported_platforms indicates those regularly tested by the development team. As a community-driven project under the GNU General Public License (GPL), the public is empowered to make small and large changes to the code base, which leads to new features, responsive bug fixes, and better usability. Blender has no price tag, but you can invest, participate, and help to advance a powerful collaborative tool: Blender is your own 3D software.

4.2 BLENDER'S DEVELOPMENT

Blender is being actively developed by hundreds of people from all around the world. These include animators, artists, VFX experts, hobbyists, scientists, and much more. All of them are united by an interest to further a completely free and open source 3D creation pipeline. The Blender Foundation supports and facilitates these goals—and employs a small staff for that—but depends fully on the global online community.

4.3 BLENDER'S MISSION

We want to build a free and open source complete 3D creation pipeline for small teams.
In this simple sentence a couple of crucial focus points come together.

- **Artists and teams:** We work for people who consider themselves artist – and who work on creating 3D individually or in small teams together. The definition for "artist" can be taken quite wide – to include engineers, product designers, architects or scientists. But each of them can be considered to have a serious interest in working with 3D software to create something related to that interest. "Blender is for artists" also means that's it not a programming API or scripting environment, these are secondary to this goal.
- **Complete 3D creation:** Blender should work for making finished products, without requirement to purchase or run other programs. Its output should satisfy the users sufficiently to share their work in public or market it as part of a living.
- **Pipeline:** We are aware of how CG production works (for animation, film, games) and we want Blender to work sufficiently in each and every aspect of such creation pipelines. This to make complex creations possible and to enable people working together.
- **Free and open source:** And not only should Blender be a complete production system, we even want this to be free and open source [6].

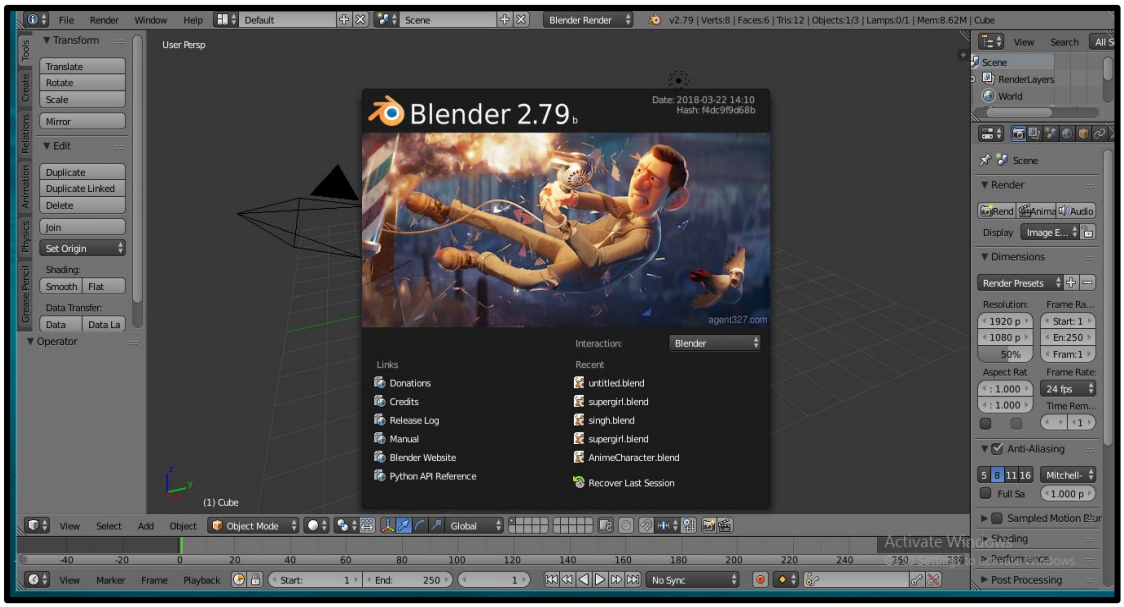


Fig 4.1 Blender's Start Window

4.4 LEARNING BLENDER INTERFACE

Blender window should look something similar to the image below. Blender's user interface is consistent across all platforms.

By default Blender starts up showing the default screen in Fig 4.2, which is separated into five areas containing the Editors listed below:

The Info Editor at the top.

- A large 3D View.
- A Timeline at the bottom.
- An Outliner at the top right.
- A Properties Editor at the bottom right.

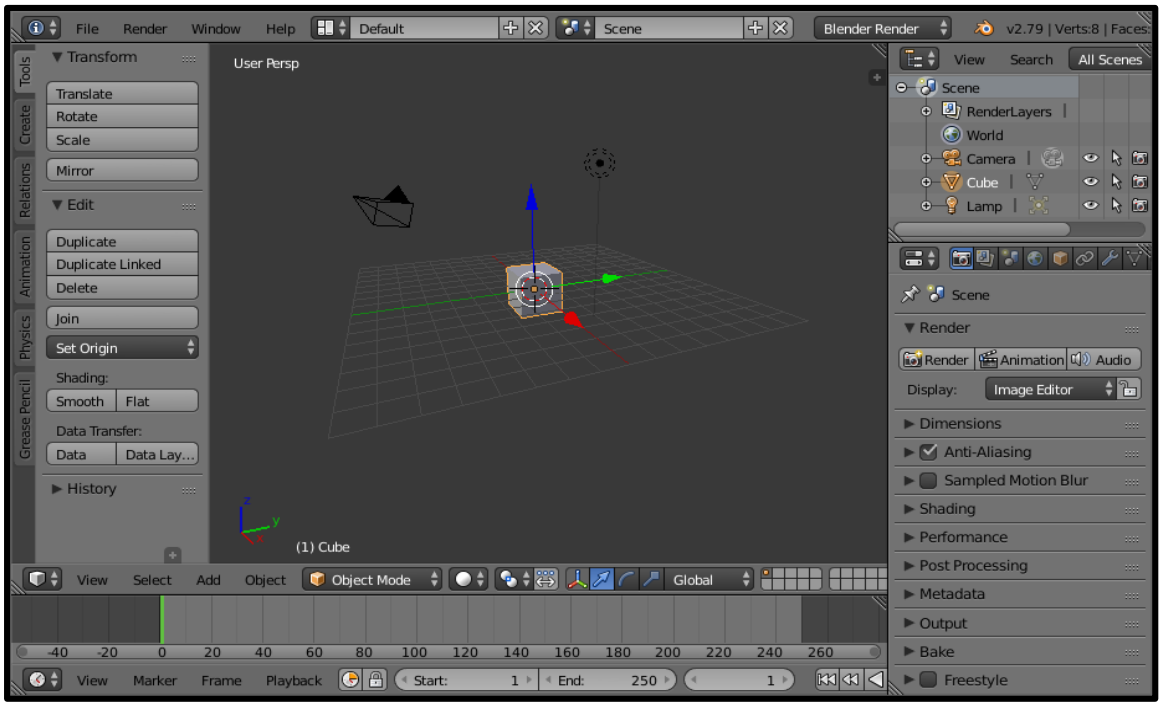


Fig 4.2 Default Blender's Window

4.4.1 Components of an Editor

In general an editor provides a way to view and modify your work through a specific part of Blender. Editors are divided into Regions. Regions can have smaller structuring elements like tabs and panels with buttons, controls and widgets placed within them.

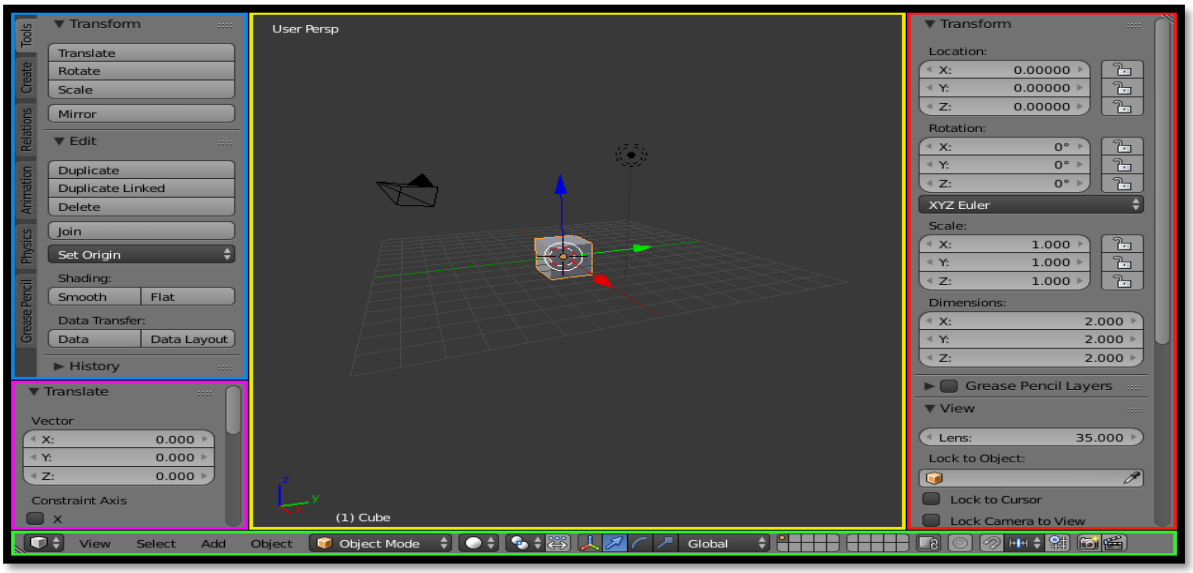


Fig 4.3 Components of an Editor

In 3-D view editor, the colourful lines indicates in Fig 4.3:

- Yellow: Main Region
- Green: Header
- Blue: Tool Shelf
- Purple: Operator Panel
- Red: Properties Region

4.5 CREATE CHARACTERS ON BLENDER

Blender helps in building supports the entirety of the 3D pipeline—modelling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation.

We create characters on Blender for our 3-D game as shown in Fig 4.4. To build own game characters in Blender, discover how to bring the character into Unity and set-up a character controller to move the character around.

- 3D Model and Sculpt a Game Character in Blender
- Use the Blender Retopology Tools
- Create UV Maps and Bake Texture Maps in Blender
- Use Texture Painting Tools in Blender
- Rig a 3D Character with the Rigify Add-on in Blender
- Animate Game Cycles for a Game Character in Blender
- Import a Blender 3D Character to Unity
- Set-up a Character Controller in Unity

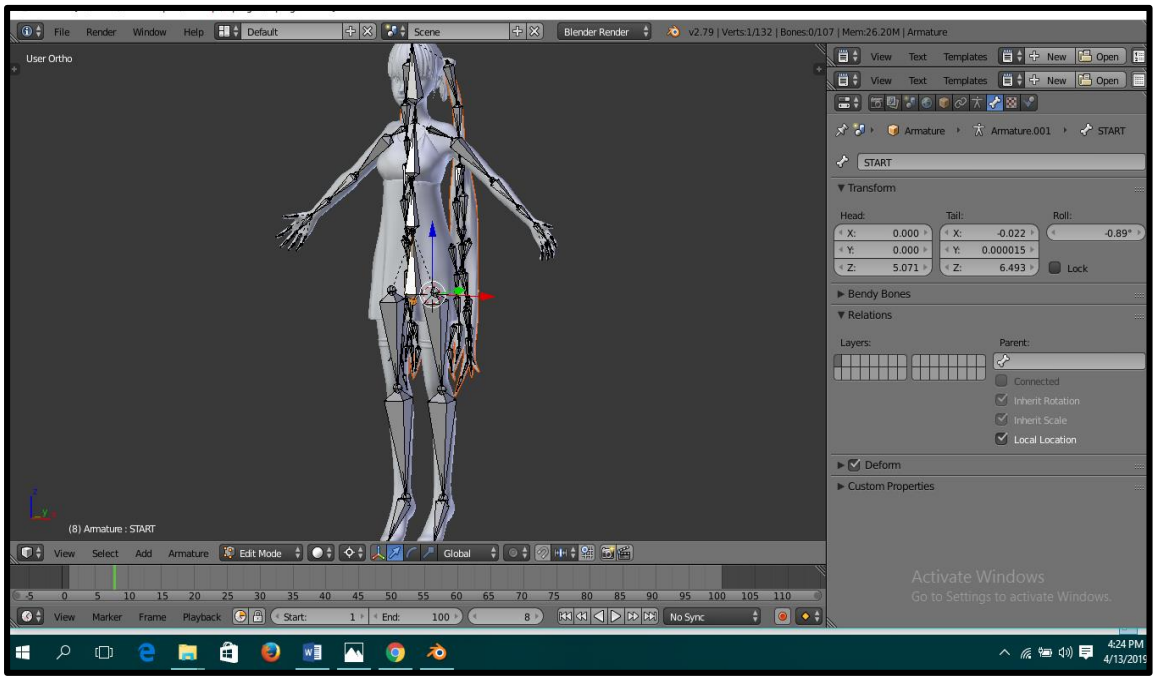


Fig 4.4 Character Build up

CHAPTER 5

PROJECT PLAN

5.1 DEVELOPMENT PROCESS

Even if one can prepare a well-thought-out design document, some features do not give the same effect in gameplay as on the paper. During implementation phase many features are added, removed or modified. Accordingly, one needs to make some modifications on game design and requirements analysis. This, one of the most suitable development methodologies for the game developers have the same idea about the advantages of iterative development process for game developers. Also, it is said that, “The more times you iterate, the better your final game will be”. We prefer the iterative development process as in fig 5.1, with specified milestones and deliverables for our project.

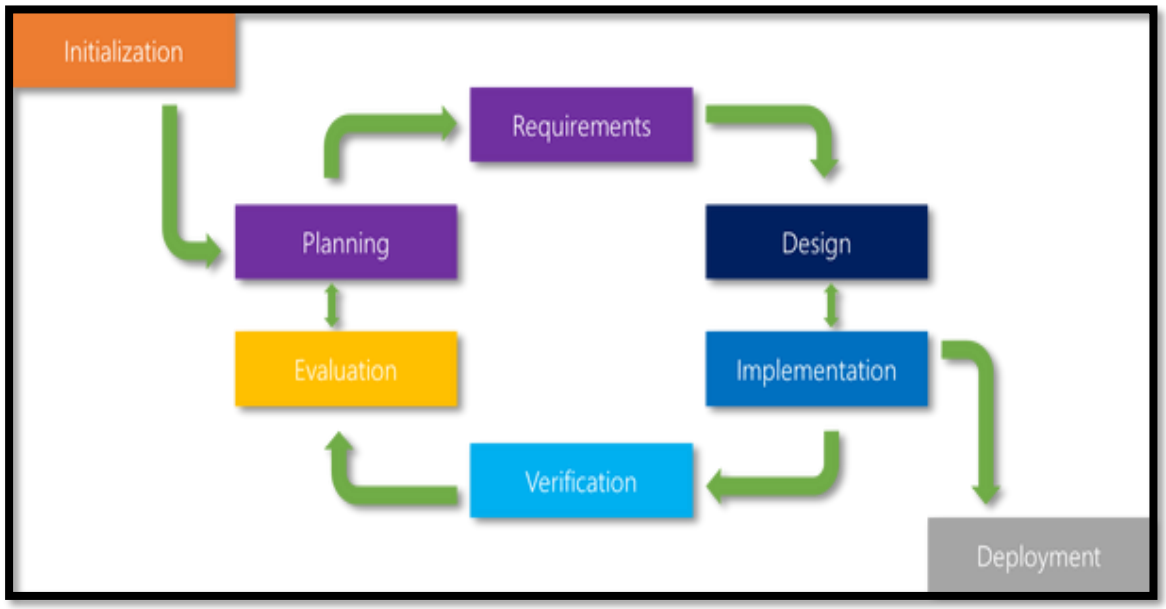


Fig 5.1 Development Process

5.2 UNDERSTANDING VIEWS OF MODEL

Everything in 3ds Max is located in a three-dimensional world. There is a variety of options for viewing this enormous stage-like space, from the tiniest details to the full extent of the scene. Using the view options discussed in this section one can move from one view to another, as the imagination and work requires. Viewports are openings into the three dimensional space of the scene like buildings looking into an enclosed trees or roads. But viewports are more than passive observation points. While creating a scene, one can use them as dynamic and flexible tools to understand 3-D relationships among objects.

5.2.1 Perspective View

Perspective view is similar to what we see with one eye closed. In this view objects look smaller as they go far away from camera/eye. Pixels covered by an object/cube will decrease as it goes far from the camera. The two most characteristic features of perspective are that objects are smaller as their distance from the observer increases; and that they are subject to foreshortening, meaning that an object's dimensions along the line of sight are shorter than its dimensions across the line of sight as shown in Fig 5.2. It Basically Means That Representation of a flat surface (such as paper), of an image as it is seen by the “eye”.

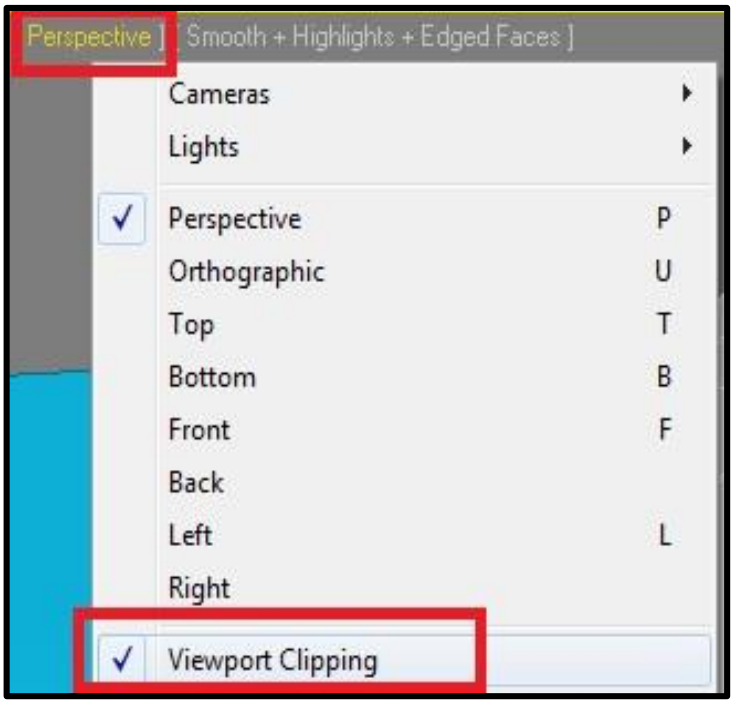


Fig 5.2 Perspective View

5.2.2 Standard Navigation Control

The standard navigation controls present in Unity which helps in controlling the image are as shown in Fig 5.3:

- Zoom Tool
- Zoom Extents
- Zoom all
- Zoom extent all
- Region zoom
- Minimum/maximum toggle
- Pan tool
- Orbit

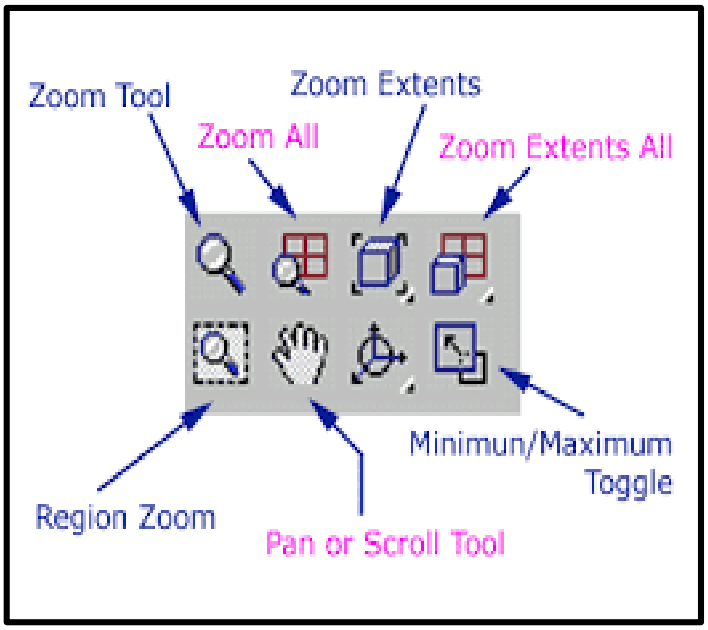


Fig 5.3 Navigation Controls

5.3 LIGHTS AND CAMERAS

Lights and Cameras are scene objects that simulate the real world counterparts. Light provides illumination for the geometry of a scene. Cameras frame the scene, providing a controllable point of view.

Cameras in Unity are used to display the game world to the player. You will always have at least one camera in a scene, but you can have more than one. Multiple cameras can give you a two-player split screen or create advanced custom effects. You can animate cameras, or control them with physics as shown in Fig 5.4. Practically anything you can imagine is possible with cameras, and you can use typical or unique cameras to fit your game's style. Cameras are essential for displaying your game to the player. They can be customized, scripted, or parented to achieve just about any kind of effect imaginable. For a puzzle game, you might keep the Camera static for a full view of the puzzle. For a first-person shooter, you would parent the Camera to the player character, and place it at the character's eye level. For a racing game, you'd probably have the Camera follow your player's vehicle. Camera will render objects uniformly, with no sense of perspective. Deferred rendering is not supported in Orthographic mode. Forward rendering is always used. The viewport size of the Camera when set to Orthographic.

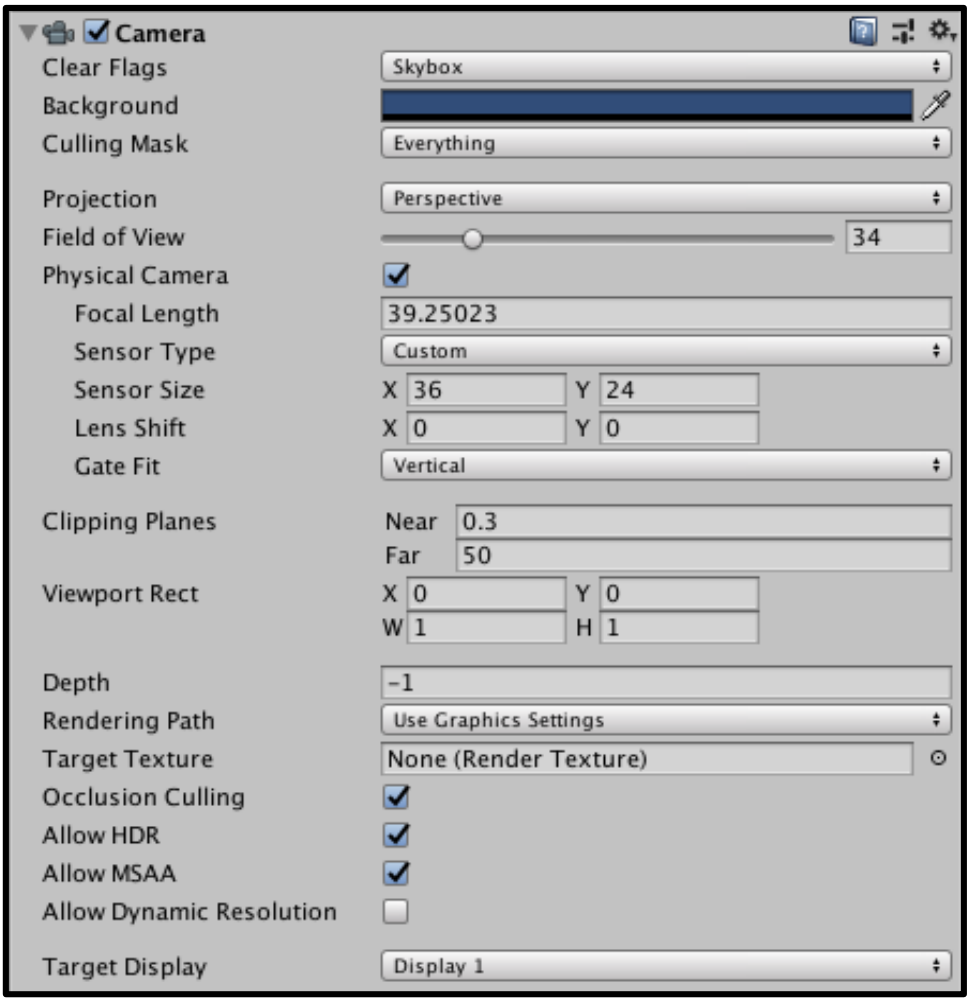


Fig 5.4 Camera Components

Lights are an essential part of every scene. While meshes and textures define the shape and look of a scene, lights define the color and mood of your 3D environment. You'll likely work with more than one light in each scene. Making them work together requires a little practice but the results can be quite amazing. Lights can be added to your scene from the GameObject->Light menu. You will choose the light format that you want from the sub-menu that appears. Once a light has been added, you can manipulate it like any other GameObject. Additionally, you can add a Light Component to any selected GameObject by using Component->Rendering->Light.

There are many different options within the Light Component in the Inspector as shown in Fig 5.5.

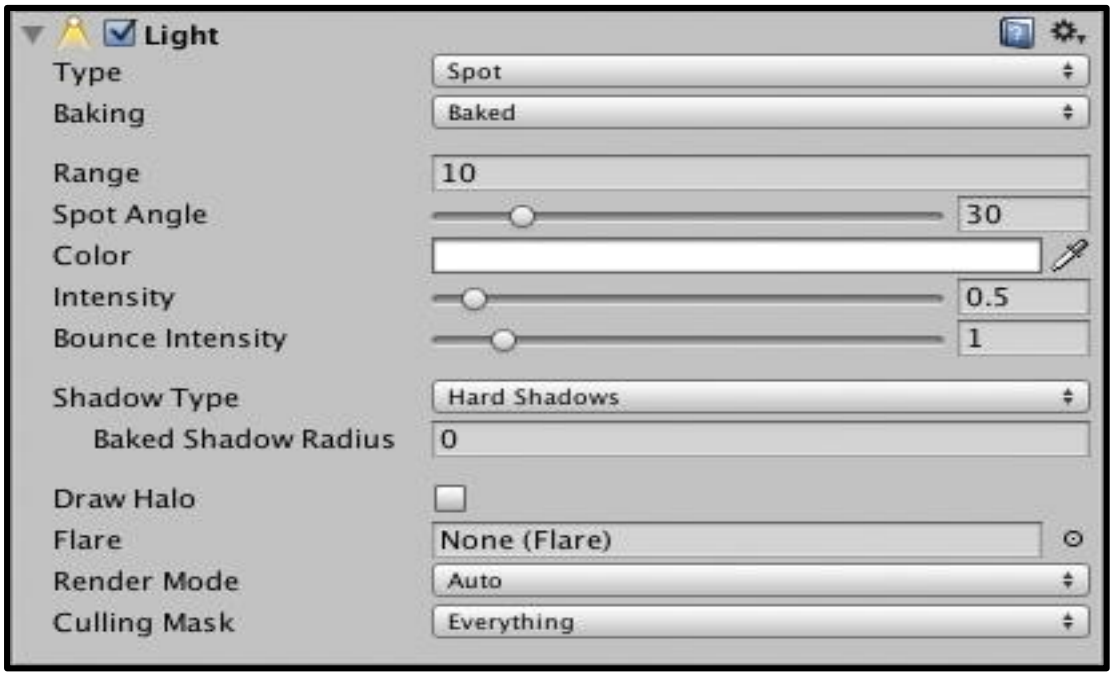


Fig 5.5 Lights Components

Photometric Light: Photometric lights use photometric values that enables one to more accurately define lights as they would be in the real world. They help to set their distribution, intensity, color, temperature and other characteristics of real world lights.

Standard Light: Standard lights are computer-based objects that simulate lights such as the light used in environment. Unlike photometric lights, Standard lights do not have physically-based intensity values. Standard lights are used in the game as this 3-d game is computer or system based game as shown in Fig 5.6 and Fig 5.7.

Fig 5.6 Dark Mode Effect

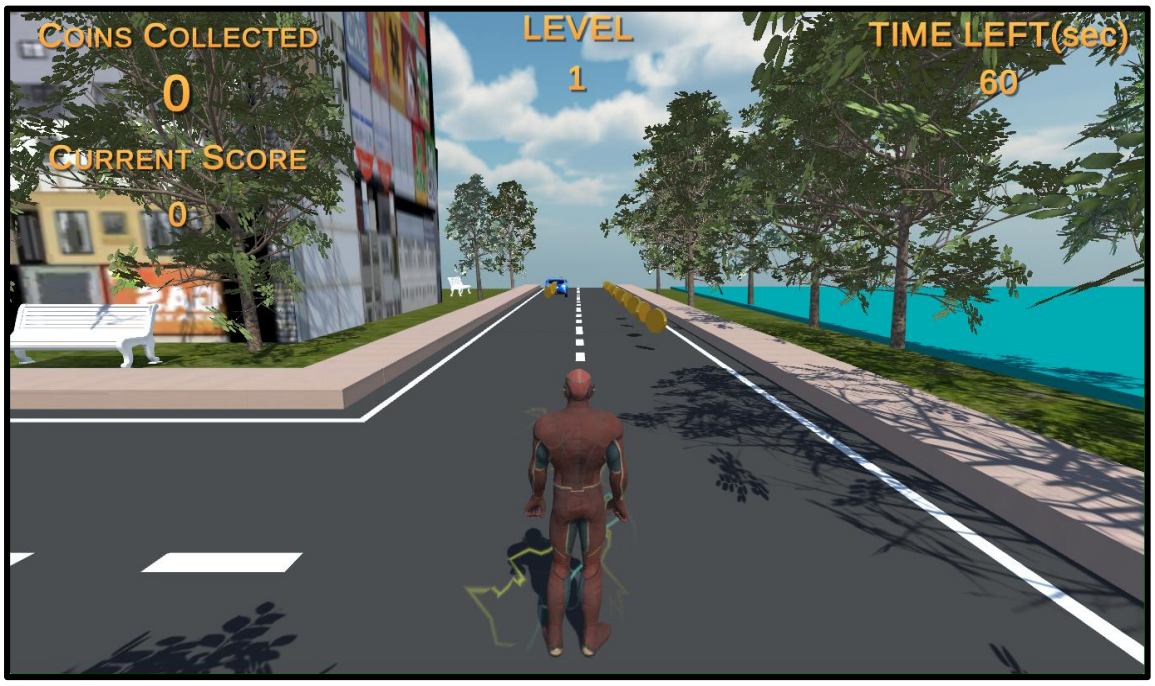


Fig 5.7 Light Mode Effect

CHAPTER 6

PRODUCTION OF GAME ASSETS (MODELLING)

6.1 DEVELOPING OBJECTS

3d-Max is an object oriented program. This means that each objects in the 3d scene carries instructions that tell 3d Max what you can do with it. These instructions vary with the type of objects. Because each object can respond to different set of commands, the commands are applied by first selecting the object and then selecting the command. This is known as a noun-verb interface, because first the object (the noun) is selected and then the command (the verb).

- Objects used to create models include “Geometry” and “Shapes”.
- Objects used to set up the scene include “Lights” and “Cameras”.

6.2 CREATION OF GAME OBJECTS

GEOMETRY (Also called “Primitives”) objects include: Boxes, Cylinders, Spheres, Cones, etc. Now we will take a very small example of creating an object. Let us see how to create buildings, environment, etc. as shown in Fig 6.1 and Fig 6.2.

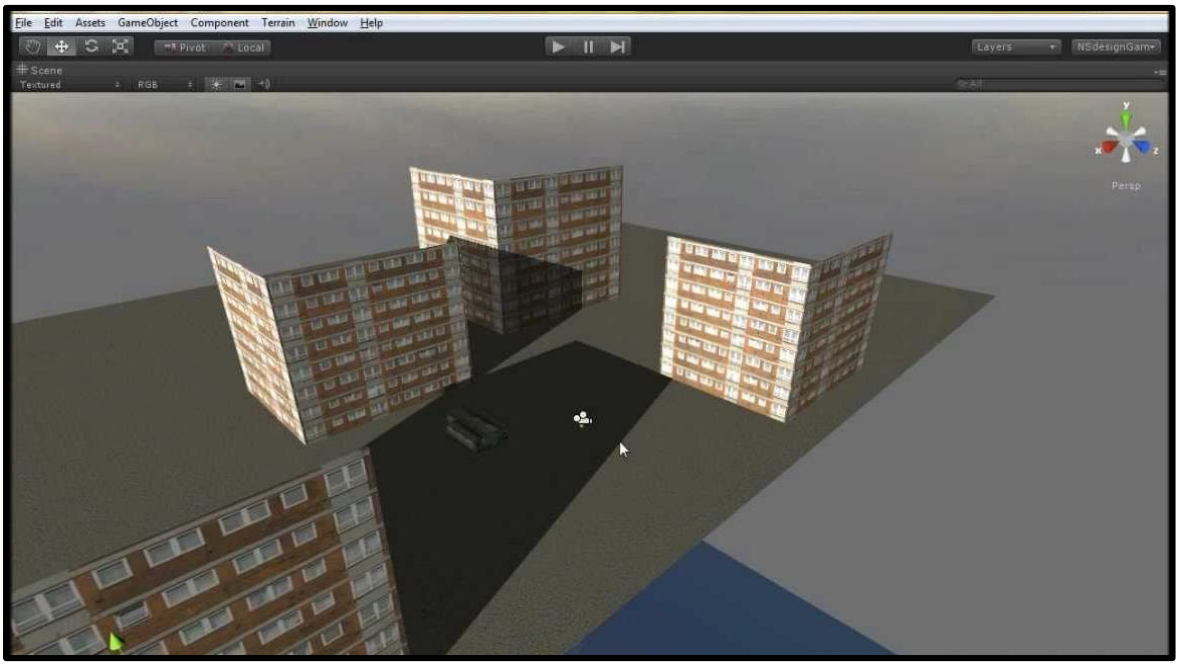


Fig 6.1 Building Creation

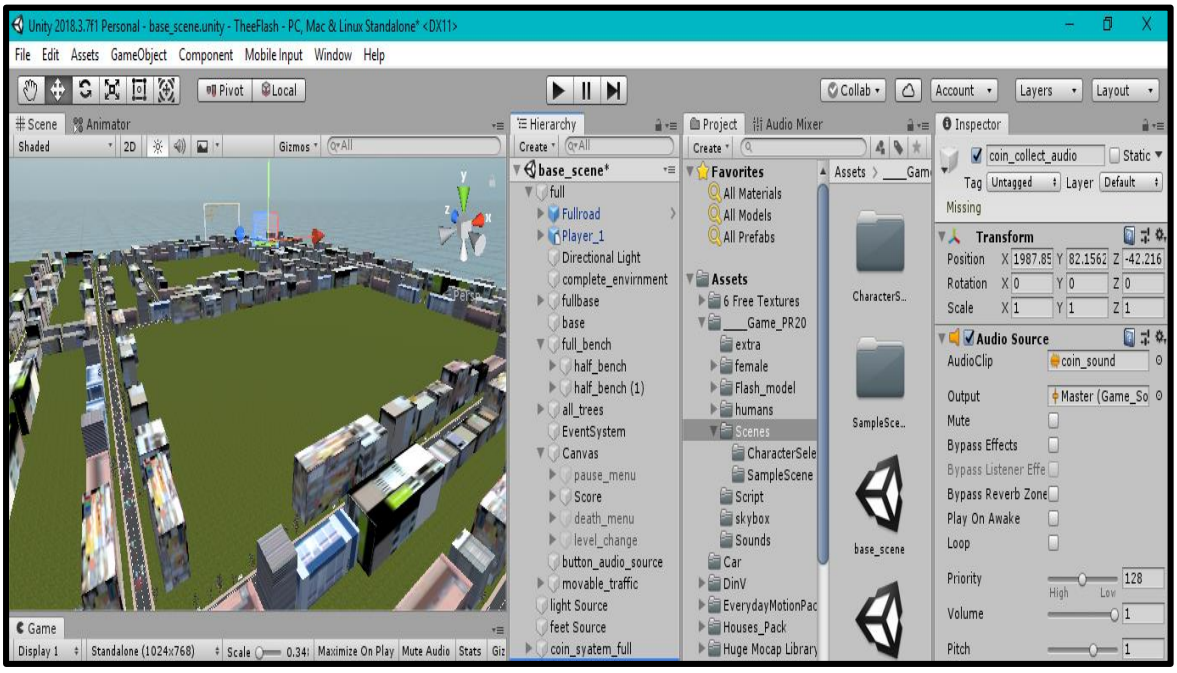


Fig 6.2 Creation of Environment

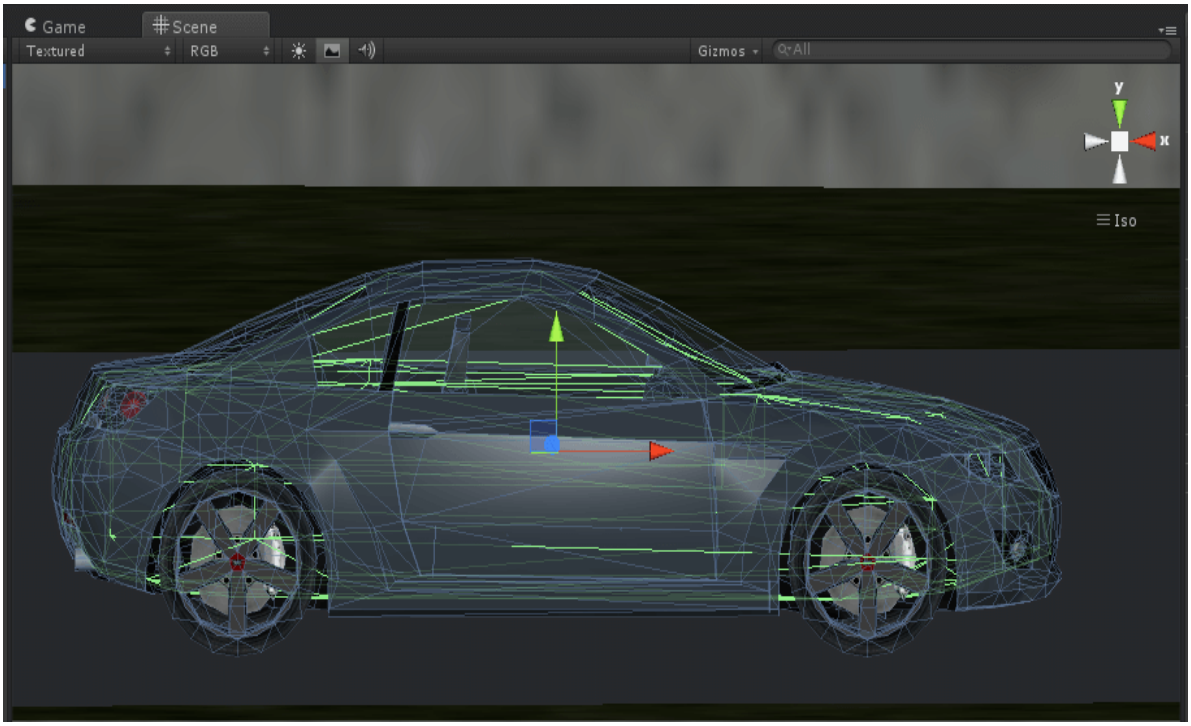


Fig 6.3 (a) Creation of Car

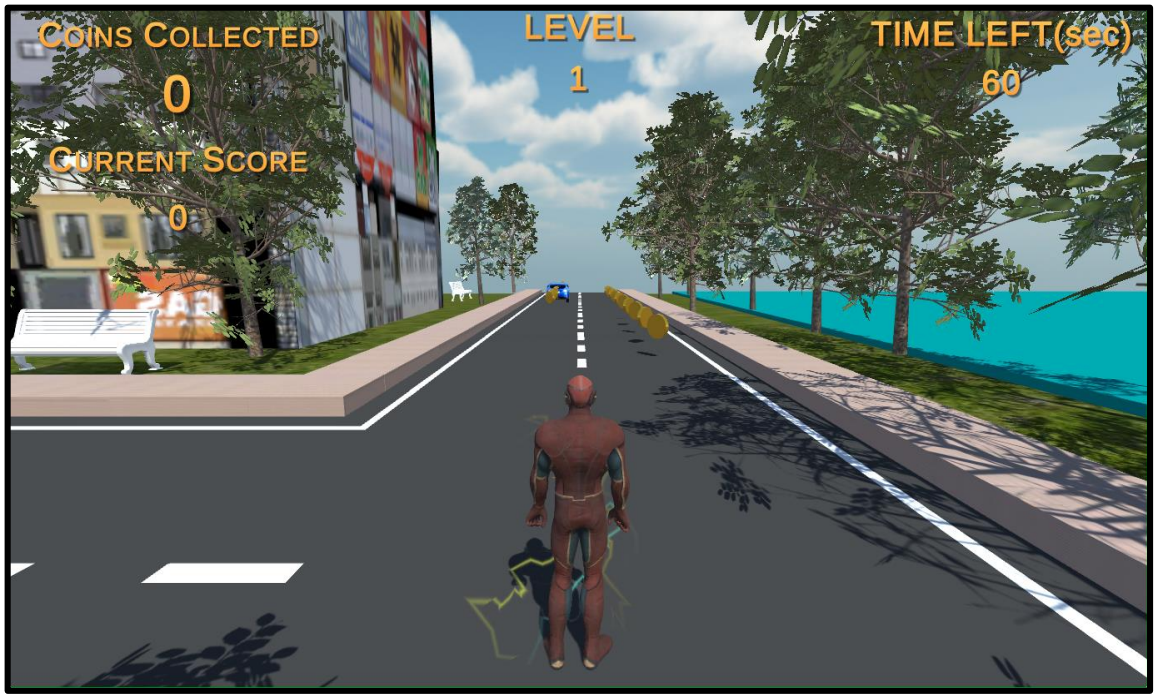


Fig 6.3 (b) Creation of Tress and Benches

To create your 3-D model or character follow these guidelines and steps:

- Name your materials, textures and geometry sensibly.
- Use normal maps for extra detail.

- Place textures in assets\object name\textures within Unity:
  It's good to have your textures set up in a pipeline that makes them convenient to work with on your project, and a good size to work with for your builds. For example, some artists generate enormous PSDs, which live outside of the Assets directory, and then from those PSDs, generate the PNGs that reside in the Assets directory in the project.
- Re-path textures before you export if required

1. Rig the character

- Prepare your model in a T-pose as shown in Fig 6.4.
- Depending on time and skill, you can rig your character by:
- Creating your own bone structure from scratch.
- Uploading your character to an auto rigger, such as Mixamo.
- Employing the rigging scripts that ship with most widely-used 3D modeling software, e.g. Biped or CAT in 3DS Max, Human IK in Maya, or Rigify in Blender.
- If you create your own custom bone structure:
- Understand the concepts of mirroring, negative scaling, and resetting transforms.
- Create the bones one by one in place for your skeleton.
- Hip bones should be the root element in the bone hierarchy.
- For both humanoid and generic rigs, a minimum of 15 bones are required.
- Follow a sensible naming/hierarchy structure.

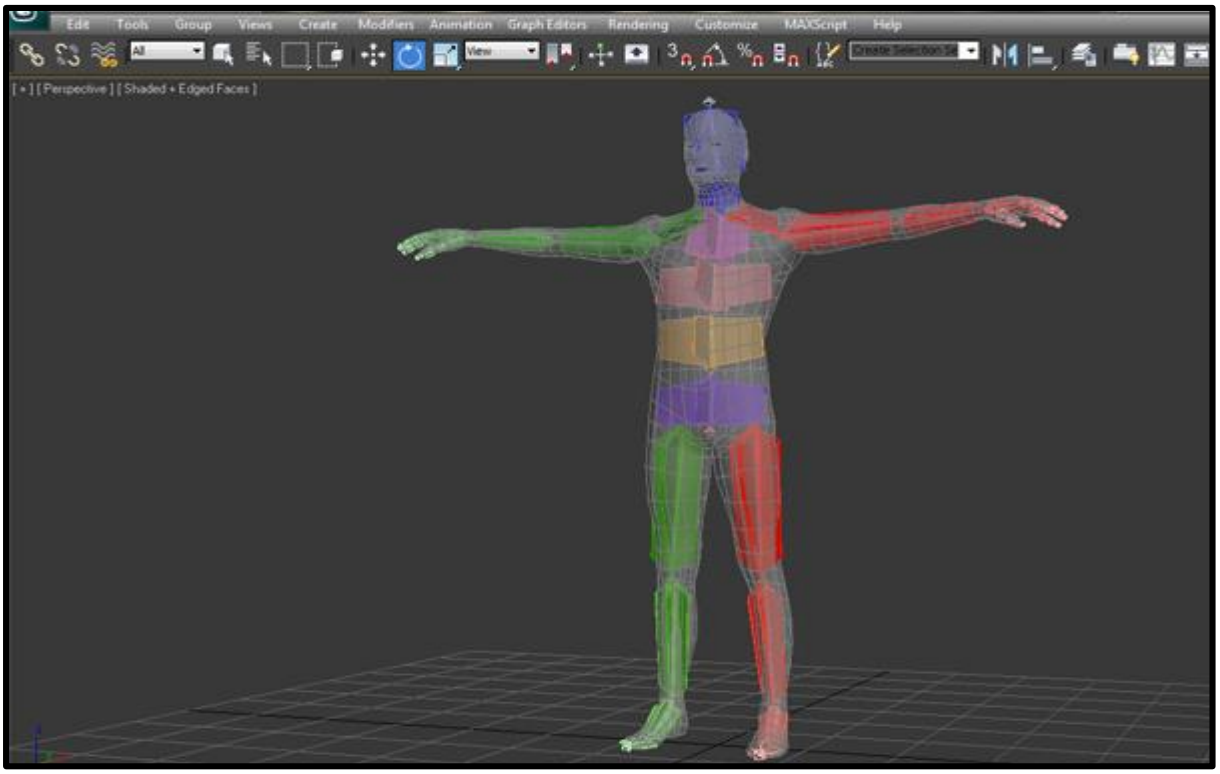


Fig 6.4 Creation of Male Character

2. Verify and Export

- Check that your character is ready to export to Unity.
- Use the FBX file format to allow for file portability and simplicity. Unity supports many native file formats, such as .max or .mb, but FBX does not require the DCC software to be installed on other collaborator's machines.
- Export, then re-import:
- Take the character that you've exported, and re-import it back into your DCC to verify that it's working the way you expect it to. Check that all your settings are correct.
- Remove unused meshes and extraneous assets like lights or cameras from your scene.
- If you have your own animation clips, check the animation box in the export dialogue.

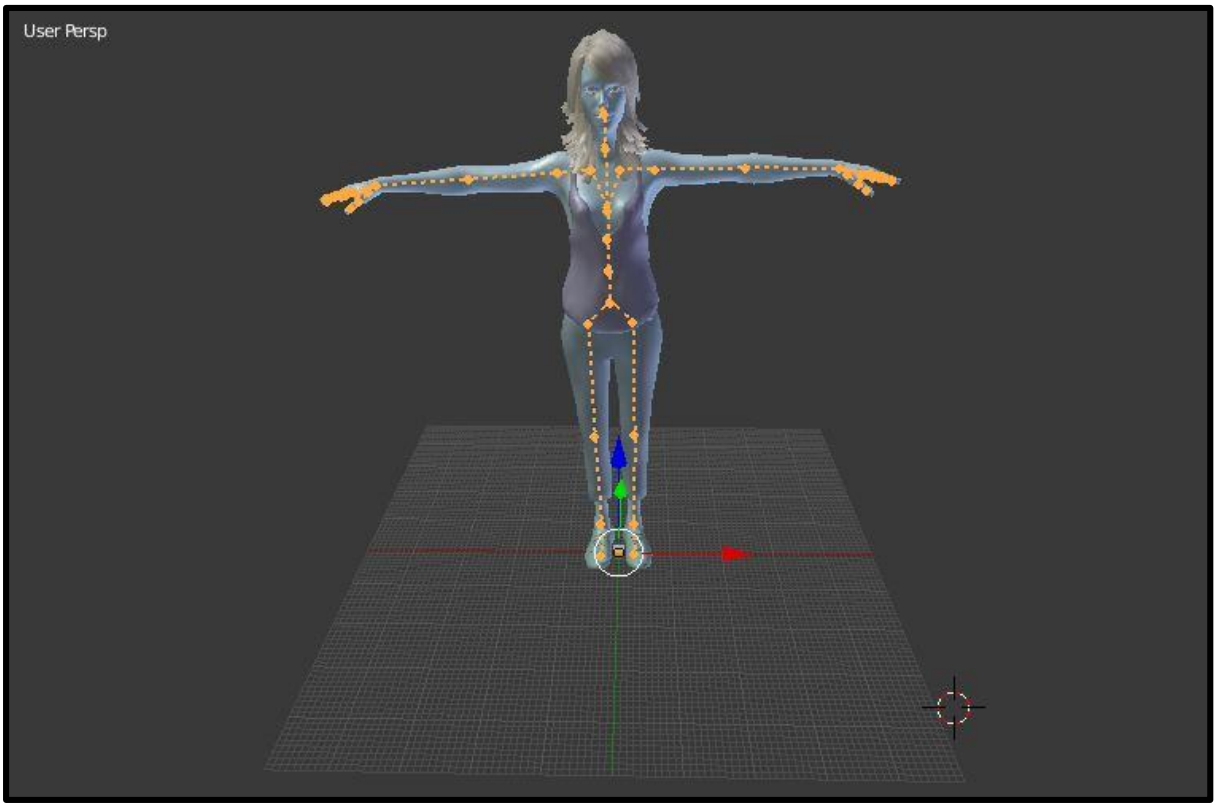


Fig 6.5 Creation of Female Character

3. Import the model

- Drag your FBX file into the Project window (if you've already exported it to here, the model will be picked up automatically).
- Select your model in the Project window and set up the options in the inspector panel via the Model, Rig and Animations tabs:

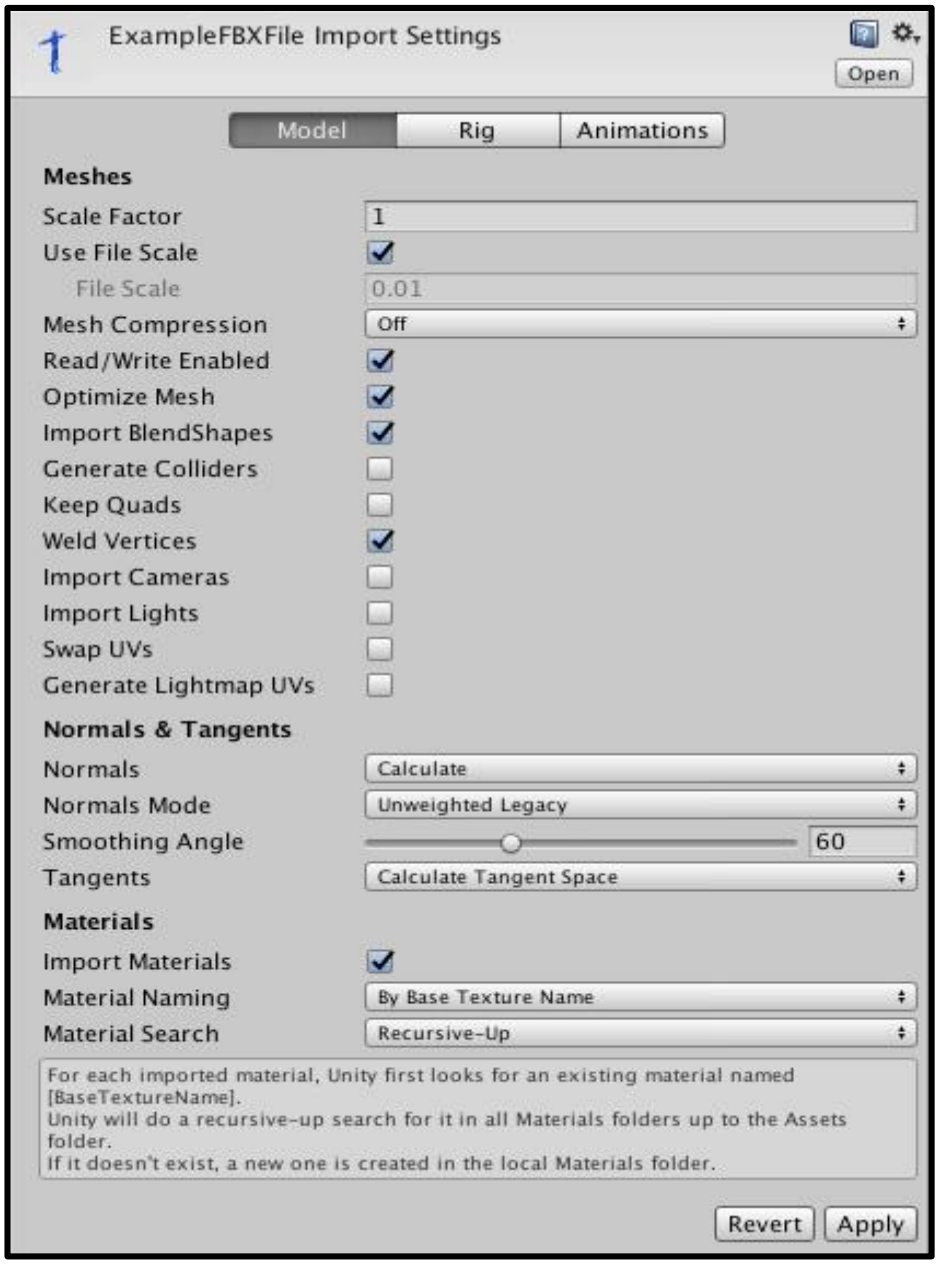


Fig 6.6 Options for Model Tab

Fig 6.7 Options for Rig Tab

Fig 6.8 Options for Animation Tab

- Check the Scale Factor; scale varies depending on units used in your DCC and export settings. Check Import Blend Shapes.
- Click apply, and drag the model into the Scene/Hierarchy from options as shown in Fig 6.6, Fig 6.7 and Fig 6.8.

4. Set up your materials

- Select your character in your scene and observe the associated materials in the Inspector, these should have been created in the Materials folder where your model is exported.
- Each Material has a drop down for Shaders; the Standard Shaders is the default selection.
- Populate slots with missing Textures.
- Adjust parameters for the Material as shown in Fig 6.9.

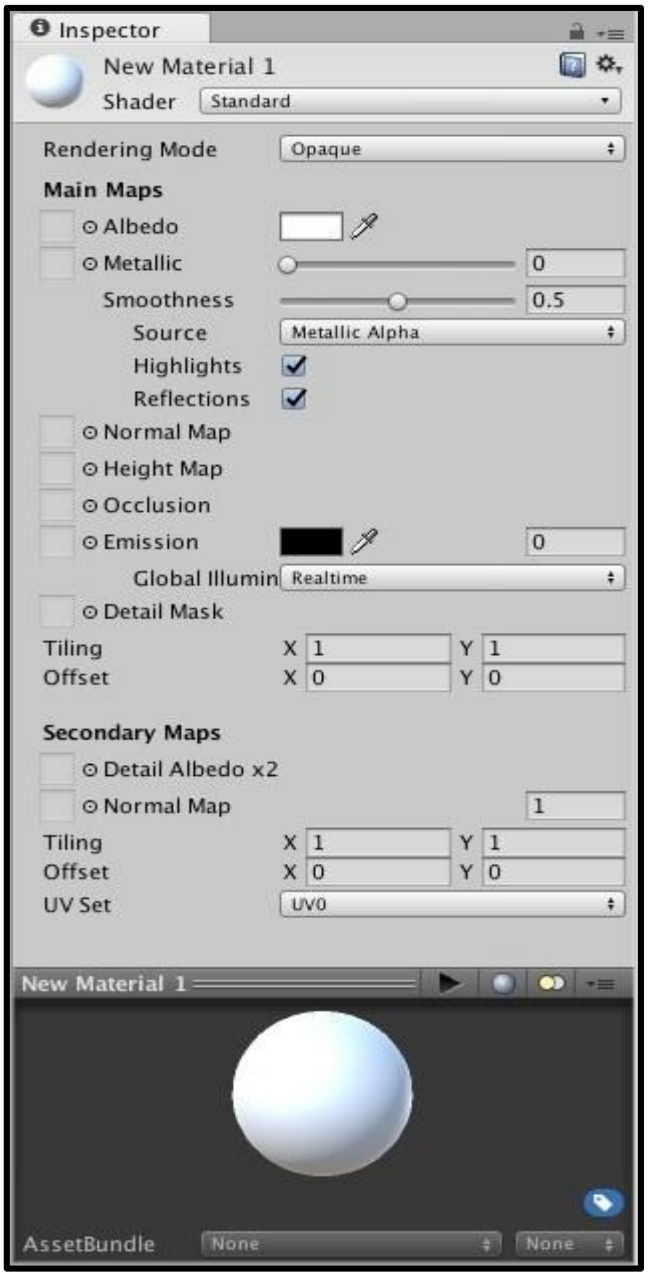


Fig 6.9 The Material parameters

5. Create an Avatar

- In this example, we'll focus on setting up a humanoid avatar.
- Unity maps your skeleton to an avatar to use with any humanoid animation: this all happens in the Rig tab.
- Select the character model FBX file in your Project window, then click on Rig tab.
- Choose Humanoid for Animation Type, then click Apply.
- Select Configure to observe skeleton mapping and muscles.

- When the skeleton turns green that means it is good to go; otherwise, assign bones to the correct slots or revisit your bone hierarchy and re-export for a closer match to avatar as shown in Fig 6.10.

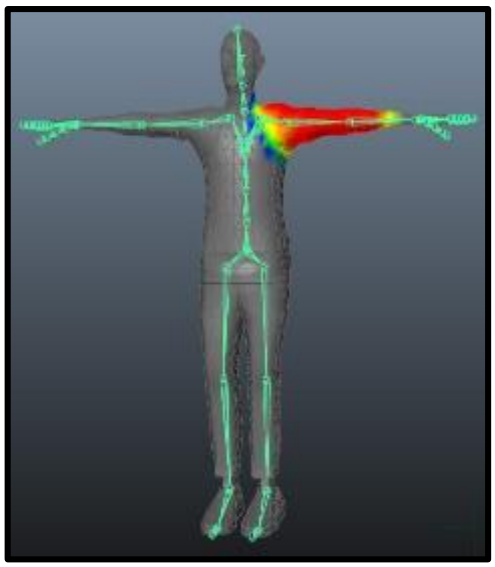


Fig 6.10 Bone Hierarchy of Human

- Test your skinning in the Muscles tab by dragging the sliders. Click done when finished.

6. Add a Character Controller and cameras:

- A fast way to do this without coding is via the Standard Assets package as shown in Fig 6.11 that comes with Unity and includes a character controller.
- For cameras, you can download Cinemachine from the Asset Store: this is Unity's new unified, procedural camera system (think of it as IK for cameras) that makes it easy and natural to craft shots, build camera rigs, tune and save properties in play mode.

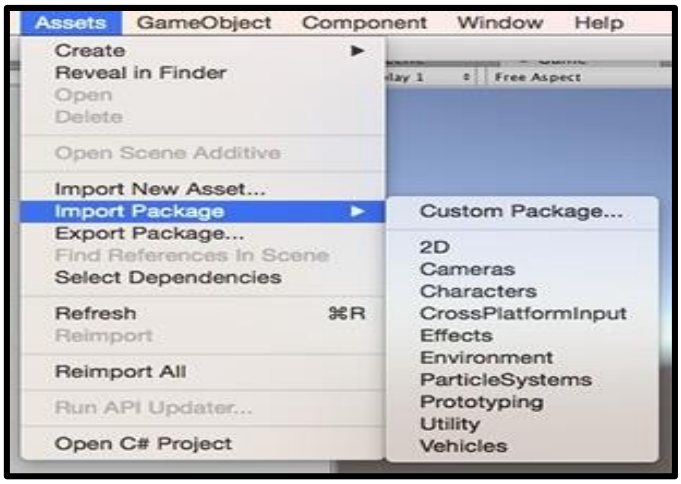


Fig 6.11 Import Package Menu

7. Add your animations

- Set up your layout to suit the workflow of adding animation to your character.
- Select your character root node and open the Animator from the Window menu pane that manages states of your character, e.g., which animation to play.
- Double-click Grounded state to open a Blend Tree for when the character is on the ground as shown in Fig 6.12.

Fig 6.12 Inspector Panel

- Apply Humanoid to the animation and choose the existing Avatar or Create from this model to retarget an animation from a different skeleton.
- Apply loop time, loop pose, bake into pose Transforms and based upon feet.

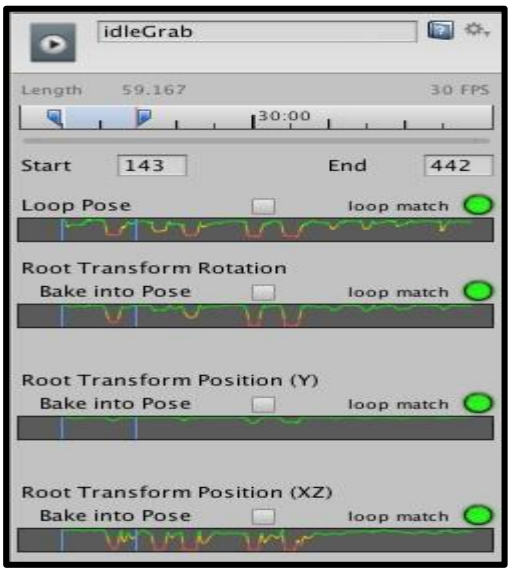


Fig 6.13 Animation Panel

- Select the Blend Tree in the Inspector and click the little circle next to an animation to choose another.
- Press Play to preview.

8. Add Blend Shapes and tweak your character

- Create some Blend Shapes or enable a Blend Shape facial rig in your character tool.
- In Unity, create a new layer in your Animator window; set blending to additive and weight:
- Select the Body and create a new animation clip.
- Animate shape nodes.
- Copy key frames to ping pong.
- Drag in your clip from the project window into the Animator on new layer.

9. Add environment elements, lights and special effects

- To test how natural-looking your character's movements are, put together a simple test environment. Use planes and primitives, levels from the sample assets, the Asset Store, and/or if available, your own artwork.
- You can automatically generate colliders from your imported meshes, however this is often inefficient. Using primitives with disabled mesh

renderers is a quick and easy way to add simple and performant collisions to your scene.

- Set Camera target.
- Adjust your free look cameras to match the environment size.
- Add post-processing profile to your camera. Adjust post-processing effects to suit your environment as shown in Fig 6.14(a) and Fig 6.14(b).

Fig 6.14(a) Final Environment with Male character

Fig 6.14(b) Final Environment with Female character

CHAPTER 7

UVW MAPPING, TEXTURING AND ANIMATION

7.1 UVW MAPPING

UVW mapping is a mathematical technique for coordinate mapping. The UVW mapping is suitable for painting an object's surface based on a solid texture. This is useful for showing a vase carved out of the marble rock. "UVW", like the standard Cartesian coordinate system, has three dimensions; the third dimension allows texture maps to wrap in complex ways onto irregular surfaces. Each point in a UVW map corresponds to a point on the surface of the object. The graphic designer or programmer generates the specific mathematical function to implement the map, so that points on the texture are assigned to (XYZ) points on the target surface. Generally speaking, the more orderly the unwrapped polygons are, the easier it is for the texture artist to paint features onto the texture. Once the texture is finished, all that has to be done is to wrap the UVW map back onto the object, projecting the texture in a way that is far more flexible and advanced, preventing graphic artifacts that accompany more simplistic texture mappings such as planar projection. For this reason, UVW mapping is commonly used to texture map non-platonic solids, non-geometric primitives, and other irregularly shaped objects, such as characters and environment [4].

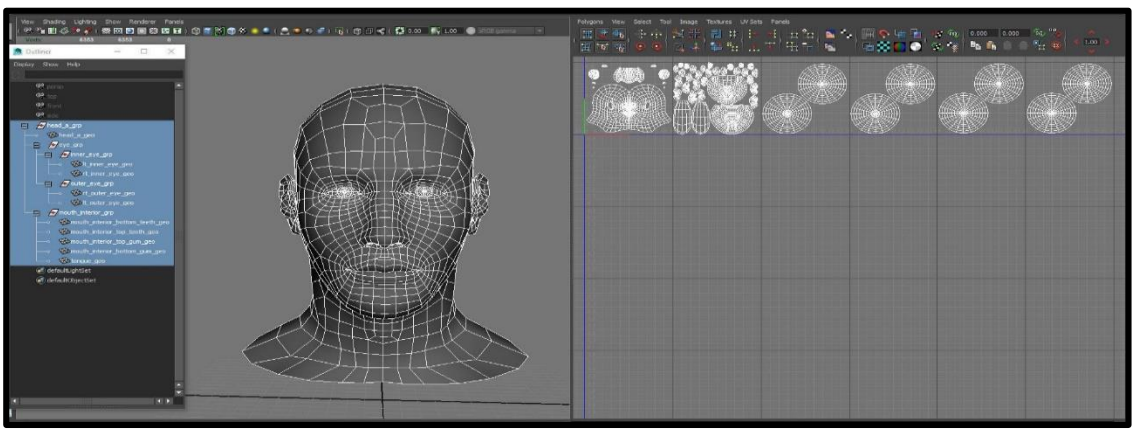


Fig 7.1 UVW Mapping of Humanoid

Without UVW coordinates, there is no way to scale the material on the object, much less accurately portray a material. The following image as shown in Fig 7.1 has UVW mapping coordinates only via the automatic generation of mapping coordinates at an object's creation. Each object receives the exact same material: Brick. The problem is that on each surface, the brick is a different size. The brick should not change size from one surface to another - it's a standard size modular unit.

Let’s place a UVW modifier on both objects.

- In the Modify Panel, drop down the list of modifiers and select UVW Map.
- Place a UVW map on each object.
- In the parameters, we have several options on the Mapping "Shape", "Size", and "Tiling". Let's start by selecting a shape that's closest to the shape of our object - the box.
- We are going to size the UVW map -- this will change the size of each brick. Think of the plane of the UVW map's gizmo being the image you are mapping onto the object. As you make the box smaller, the image is getting smaller. If you "squash" the box, you are "squashing" the image to be mapped.
- UVW mapping deals with the scale and positioning of a material onto an object. It is a "projector" helper object, in that it takes a 2D image and projects it into 3D space onto geometry.
- If the material is set to Tile, which it does by default, the material will repeat in each direction beyond the UVW map's size. If the UVW map is set to tile, it will divide the UVW map into equal parts and tile the texture within the UVW map's size.
- If you know the size of an individual brick, and you know how many bricks are in the height direction of the image, you can be very accurate in how large the bricks should be. Let's use 2-1/2" as the height of a brick in this case. We have 24 bricks, so the total height for this tile should be 60". Therefore the size of our UVW map should be 60" x 60" x 60", since the image is also a square.

- This size will not change for any instances of this particular brick. So the modify panel for the UVW map will look the same for every object that has that particular material on it.
- Now, the brick is the same size on each object, and it also has a realistic scale for a convincing scene.
- Often, you will not need to be this accurate, even if you are able. In most cases it is an approximation.
- In addition to sizing the brick, we can also use the UVW gizmo to position the bricks on the surfaces. In the Sub-Object mode of the UVW Map, you will see Gizmo. Using the Move tool and the Transform Gizmo (the XYZ indicator on the screen), you can position the bricks on the surface as shown in Fig 7.2.

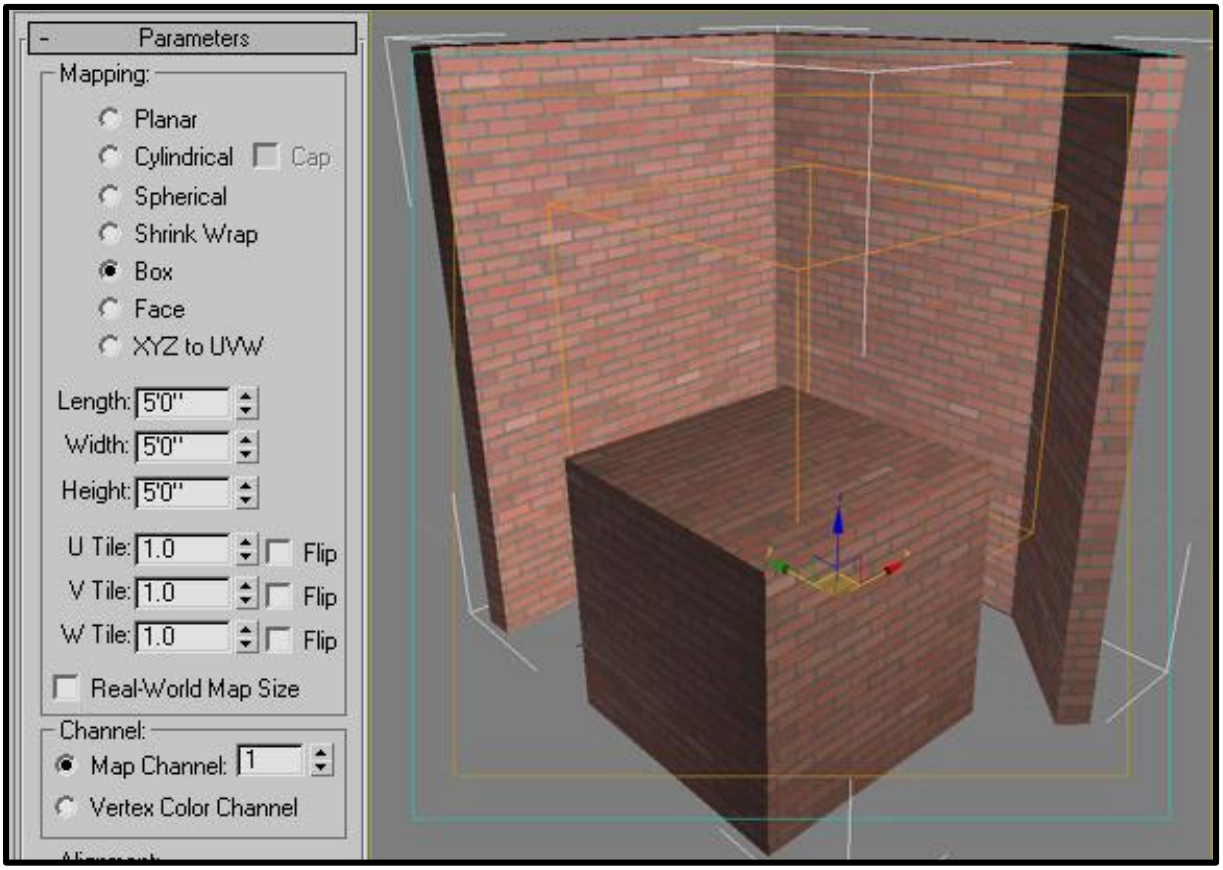


Fig 7.2 Mapping Parameters of Building wall

7.2 TEXTURING

This section covers how to export an outline of your UV map, and how to load images into the UV/Image editor.

7.2.1 Applying Textures to UVs

The UV/Image Editor allows you to map textures directly to the mesh faces. The 3D View editor shows you the object being textured. If you set this editor into textured viewport shading, you will immediately see any changes made in the UV/Image and this editor, and vice versa. You can edit and load images, and even play a game in the Blender Game Engine with UV textures for characters and object, without a material, and still see them in the 3D View. This is because no real rendering is taking place; it is all just viewport shading. If you were to apply an image to UVs then render, the texture would not show up by default.

To render an image however, you must:

- Create a Material for the object, and
- Tell Blender to use the UV textures on faces when rendering.

To create a Material, you have to click Add New Material in the Shading context.

There are two ways to tell Blender to use the UV texture when rendering: the Proper way and the Quick Way:

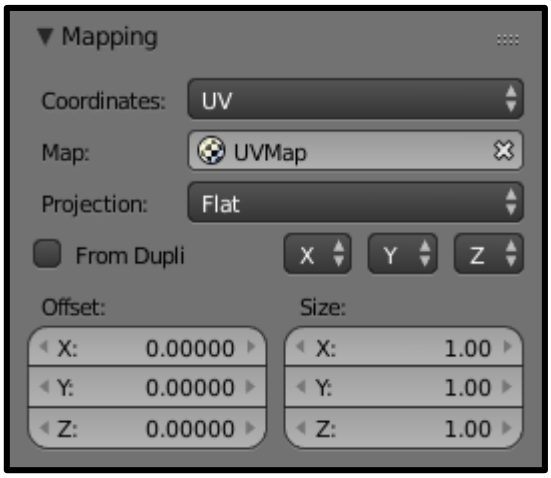


Fig 7.3 Texture setup to map using its UV coordinates

In the Texture channel panel, add a New Texture and define the texture as an image and load the image you want to use. In the Mapping section, choose UV from the Coordinates menu, and select the UV map to use as shown in Fig 7.3. Make sure it is mapped to Color in the Influence section as well (it will be mapped to Color by default, and the UV texture is named "UVTex" by default).

If the image has an alpha channel and you want to use it, click "Use Alpha" in the Map Image panel. Full details of using Image textures are on the Image Textures page.

7.2.2 Face Textures

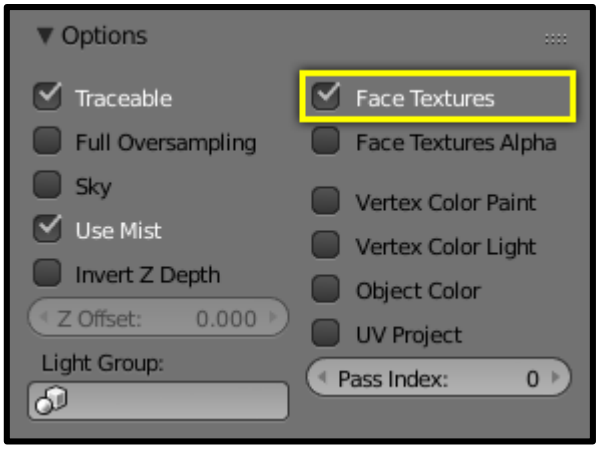


Fig 7.4 The Material panel with activated Face Textures button

An alternative way is to set up a Face Textures Material as shown in Fig 7.4. To do so, with the Properties editor displayed, press F5 to display the Shaders Buttons. In the Properties editor, Material settings, click Add New material.

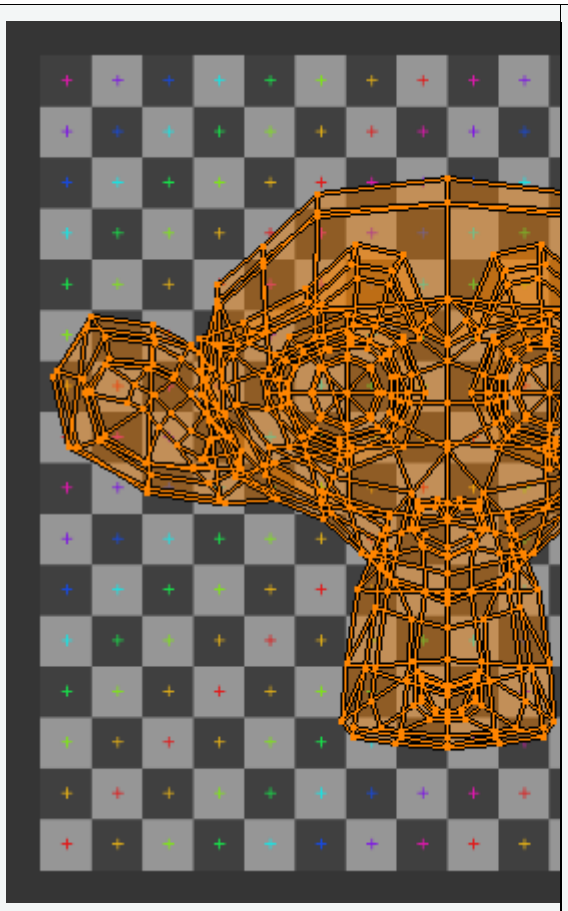
On the Options panel, enable Face Textures. This way is quick, but bypasses the normal rendering system for fast results, but results which do not respect transparency and proper shading.

7.2.2 Using the Test Grid

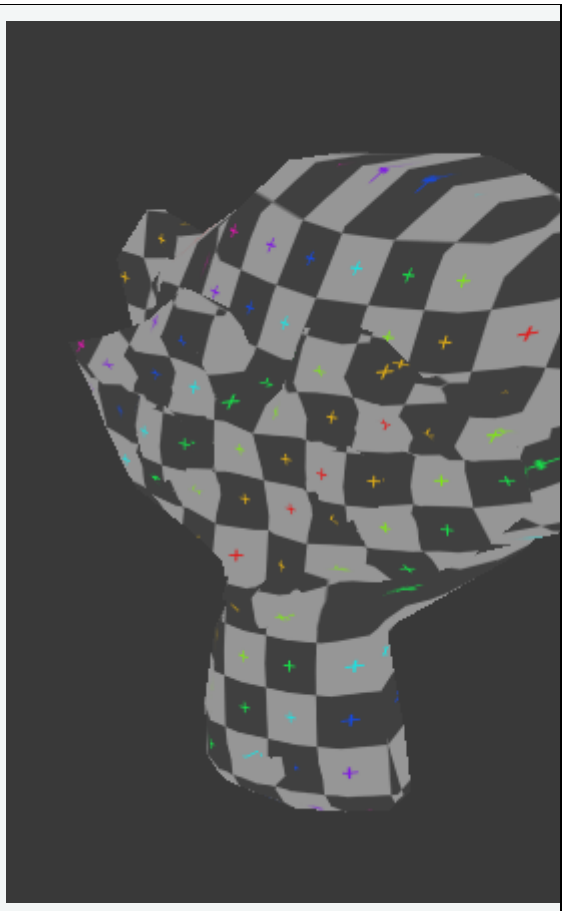
If your image is a base uniform pattern and you want the application of that image to your model to look like cloth, you do not want any stretching (unless you want the cloth to look like spandex).

7.2.3 Modifying Your Image Texture

The advantage to saving as a separate file is that you can easily switch textures just by copying other image files over it, and you can use external editing programs to work on it. The advantage of packing is that your whole project is kept in the blend-file, and that you only have to manage one file.



The test grid applied to the UVs.



A preview of the texture on the geometry.

Fig 7.5 Using Test Grid

When you render, the mesh will have the test grid as its colors, and the UV texture will be the size image you specified as shown in Fig 7.5.

7.3 ANIMATION

Animation is possibly the most difficult art form anyone can attempt. It takes years to get good at it and one is never done learning. It is also impossible to be “taught” animation. One can be shown how to make a model or do an effect but unless one practice and just go off on his own, one, will never truly know how to do animation. Much like programming and other computer related tasks, animation is “self-taught”.

Animation is making an object move or change shape over time. Objects can be animated in many ways:

- Moving as a whole object: Changing their position, orientation or size in time;
- Deforming them: Animating their vertices or control points.
- Inherited animation: Causing the object to move based on the movement of another object (e.g. its parent, hook, armature, etc.).

In this chapter, we will cover the first two, but the basics given here are actually vital for understanding the following chapters as well.

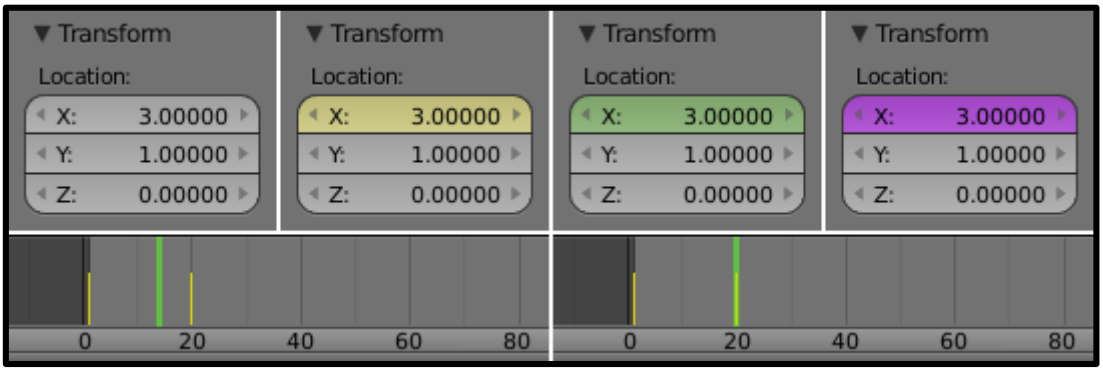


Fig 7.6 State Colors

Properties have different colors and menu items for different states as shown in Fig 7.6.

Table 7.1 Properties of state colors

Gray	Default
Yellow	Key frames
Green	Animation
Purple	Driver

Object animation is a form of stop motion animation that involves the animated movements of any non-drawn objects such as toys, blocks, dolls, etc. which are not fully malleable, such as clay or wax, and not designed to look like a recognizable human or animal character as shown in Fig 7.7. Object animation is considered a different form of animation distinct from model animation and puppet animation, as these two forms of stop-motion animation usually use recognizable characters as their subjects [5].

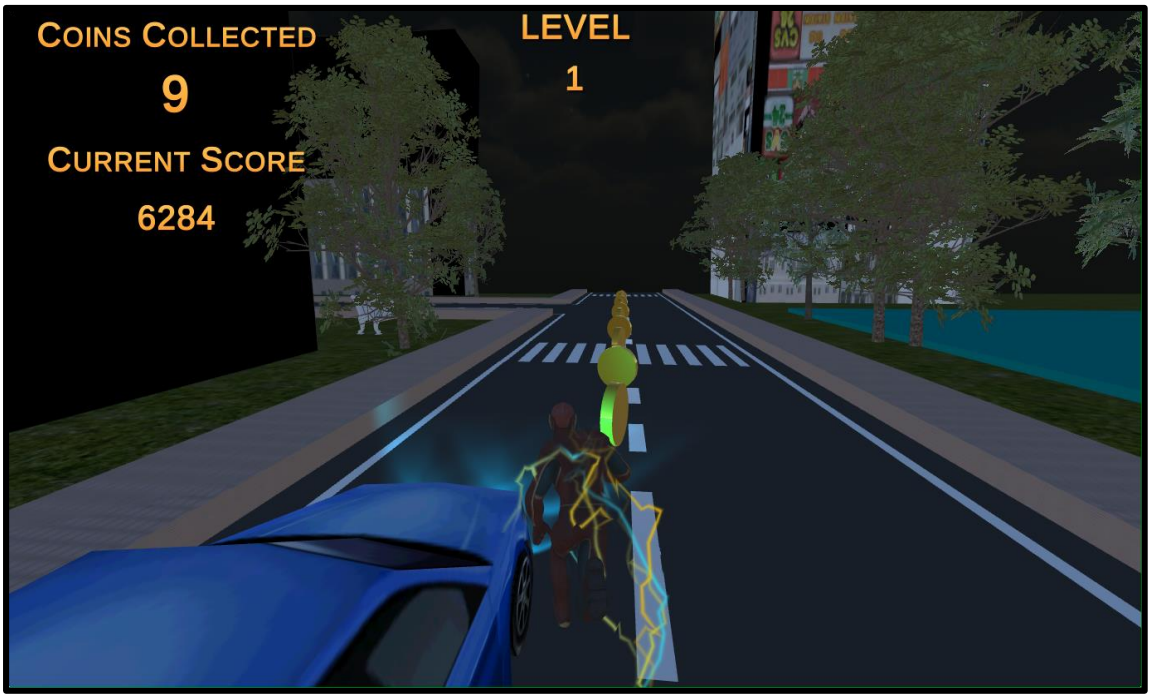


Fig 7.7 Animation

CHAPTER 8

SCRIPTING OF A GAME

8.1 INTRODUCTION

Scripting is an essential ingredient in all games. Even the simplest game needs scripts, to respond to input from the player and arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behaviour of objects or even implement a custom AI system for characters in the game.

Fig 8.1 Scripting

Scripting is a skill that takes some time and effort to learn. The intention of this section is not to teach you how to write script code from scratch, but rather to explain the main concepts that apply to scripting in Unity. Although Unity uses an implementation of

the standard Mono runtime for scripting, it still has its own practices and techniques for accessing the engine from scripts. This section explains how objects created in the Unity editor are controlled from scripts, and details the relationship between Unity's gameplay features and the Mono runtime [1].

8.1.1 Creating and Using Scripts

The behaviour of Game Object is controlled by the Components that are attached to them. Although Unity's built-in Components can be very versatile, you will soon find you need to go beyond what they can provide to implement your own gameplay features. Unity allows you to create your own Components using scripts. These allow you to trigger game events, modify Component properties over time and respond to user input in any way you like.

Unity supports the C# programming language natively. C# (pronounced C-sharp) is an industry-standard language similar to Java or C++.

Unlike most other assets, scripts are usually created within Unity directly. You can create a new script from the Create menu at the top left of the Project panel or by selecting **Assets > Create > C# Script** from the main menu.

The new script will be created in whichever folder you have selected in the Project panel. The new script file's name will be selected, prompting you to enter a new name as shown in Fig 8.2.

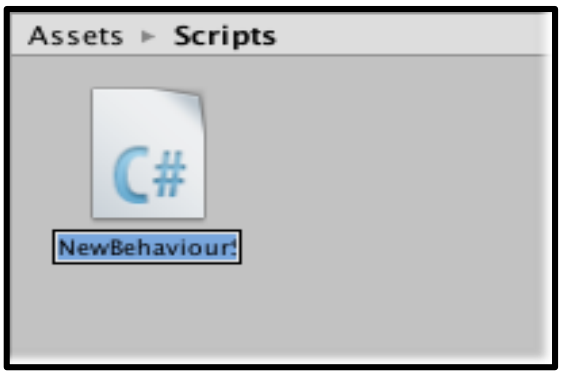


Fig 8.2 Script a Folder

A list of elements an object might have as follows:

- Transform
- Cube (Mesh Filter)
- Box Collider
- Mesh Renderer

8.2 TRANSFORM

The Transform component determines the Position**,** Rotation, and Scale of each object in the scene. Every GameObject has a Transform as shown in Fig 8.3.

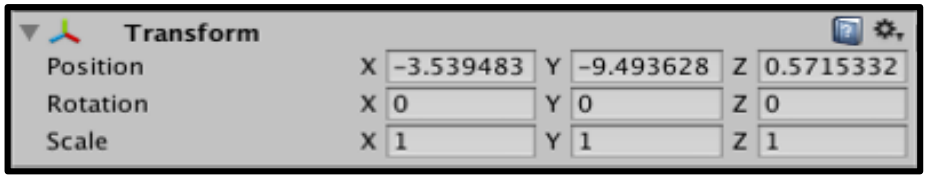


Fig 8.3 Transform Panel

Table 8.1 Transformation Properties

Property:	**Function:**
Position	Position of the Transform in X, Y, and Z coordinates.
Rotation	Rotation of the Transform around the X, Y, and Z axes, measured in degrees.
Scale	Scale of the Transform along X, Y, and Z axes. Value “1” is the original size (size at which the object was imported).

The position, rotation and scale values of a Transform are measured relative to the Transform’s parent. If the Transform has no parent, the properties are measured in world space.

8.3 RENDERER

Rendering is the process of generating an image from a 2D or 3D model (or models in what collectively could be called a scene file) by means of computer programs.

Computer programs such as Unity Game Engine, Maya, Computer Animation & Modelling Software, etc. Also, the results of such a model can be called a rendering as shown in Fig 8.4.

Rendering may be done ahead of time (pre-rendering) or it can be done in on-the-fly in real time. Real-time rendering is often used for 3-D video games, which require a high level of interactivity with the player. Pre-rendering, which is CPU-intensive but can be used to create more realistic images, is typically used for movie creation.

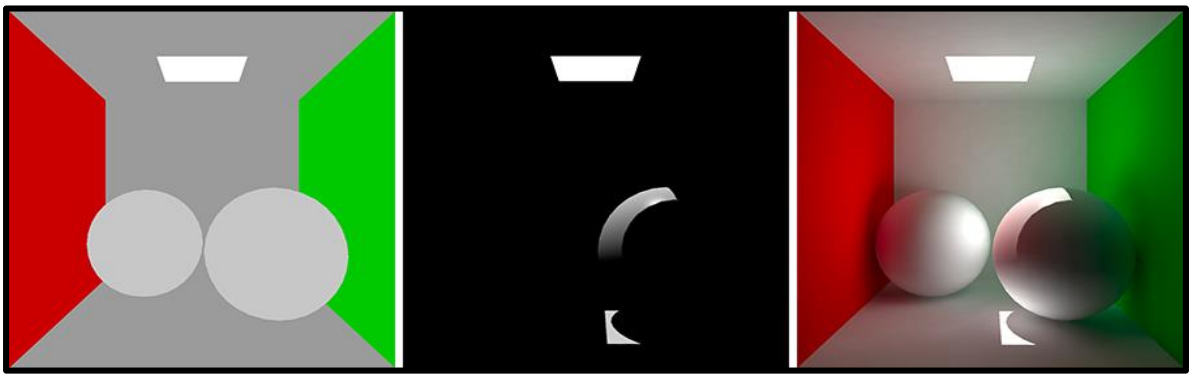


Fig 8.4 Rendering of an object

8.4 PHYSICS

To have convincing physical behavior, an object in a game must accelerate correctly and be affected by collisions, gravity and other forces. Unity's built-in physics engines provide components that handle the physical simulation for you. With just a few parameter settings, you can create objects that behave passively in a realistic way (i.e., they will be moved by collisions and falls but will not start moving by themselves). By controlling the physics from scripts, you can give an object the dynamics of a vehicle, a machine, or even a piece of fabric. This section gives an overview of the main physics components in Unity, with links for further reading.

Since a Rigid body component takes over the movement of the GameObject it is attached to, you shouldn't try to move it from a script by changing the Transform properties such as position and rotation. For example, you may want to control your character directly from script code but still allow it to be detected by triggers. This kind of non-physical motion produced from a script is known as kinematic motion. The Rigid body component has a

property called Is Kinematic which removes it from the control of the physics engine and allow it to be moved kinematically from a script. It is possible to change the value of Is Kinematic from a script to allow physics to be switched on and off for an object, but this comes with a performance overhead and should be used sparingly.

8.5 COLLIDERS

When collisions occur, the physics engine calls functions with specific names on any scripts attached to the objects involved. You can place any code you like in these functions to respond to the collision event. For example, you might play a crash sound effect when a car bumps into an obstacle.

On the first physics update where the collision is detected, the **OnCollisionEnter** function is called. During updates where contact is maintained, **OnCollisionStay** is called and finally, **OnCollisionExit** indicates that contact has been broken. Trigger colliders call the analogous **OnTriggerEnter**, **OnTriggerStay** and **OnTriggerExit** functions. Note that for 2D physics, there are equivalent functions with 2D appended to the name, e.g., OnCollisionEnter2D. Full details of these functions and code samples can be found on the Script Reference page for the **MonoBehaviour** class.

With normal, non-trigger collisions, there is an additional detail that at least one of the objects involved must have a non-kinematic Rigid body (i.e., Is Kinematic must be switched off). If both objects are kinematic Rigid bodies then **OnCollisionEnter**, etc., will not be called. With trigger collisions, this restriction doesn’t apply and so both kinematic and non-kinematic rigid bodies will prompt a call to **OnTriggerEnter** when they enter a trigger collider.

This is a GameObject with a Collider and a kinematic Rigid body attached (i.e., the Is Kinematic property of the Rigid body is enabled). You can move a kinematic rigid body object from a script by modifying its Transform Component but it will not respond to collisions and forces like a non-kinematic rigid body. Kinematic rigid bodies should be used for colliders that can be moved or disabled/enabled occasionally but that should otherwise behave like static colliders. An example of this is a sliding door that should normally act as

an immovable physical obstacle but can be opened when necessary. Unlike a static collider, a moving kinematic rigid body will apply friction to other objects and will “wake up” other rigid bodies when they make contact.

Even when immobile, kinematic rigid body colliders have different behaviour to static colliders. For example, if the collider is set to as a trigger then you also need to add a rigid body to it in order to receive trigger events in your script. If you don’t want the trigger to fall under gravity or otherwise be affected by physics then you can set the Is Kinematic property on its rigid body.

A Rigid body component can be switched between normal and kinematic behaviour at any time using the Is Kinematic property.

8.6 TIMELINE AND CORRECTION WINDOW

The Inspector window displays information about the selected GameObject including all attached components and their properties. This section documents the properties in the Inspector window that appear when you select one or many Timeline objects: a Timeline Asset, a track, or a clip.

If you select a single Timeline object, the Inspector window displays the properties for the selected object. For example, if you select an Animation clip, the Inspector window shows the common properties and playable asset properties for the selected Animation clip.

Inspector window when selecting an Animation clip in the Timeline Editor window if you select multiple Timeline objects, and the selection includes Timeline objects with common properties, the Inspector window shows two sections: a section with properties that apply to the entire selection, and a section of common properties that apply to each selected object individually as shown in Fig 8.5.

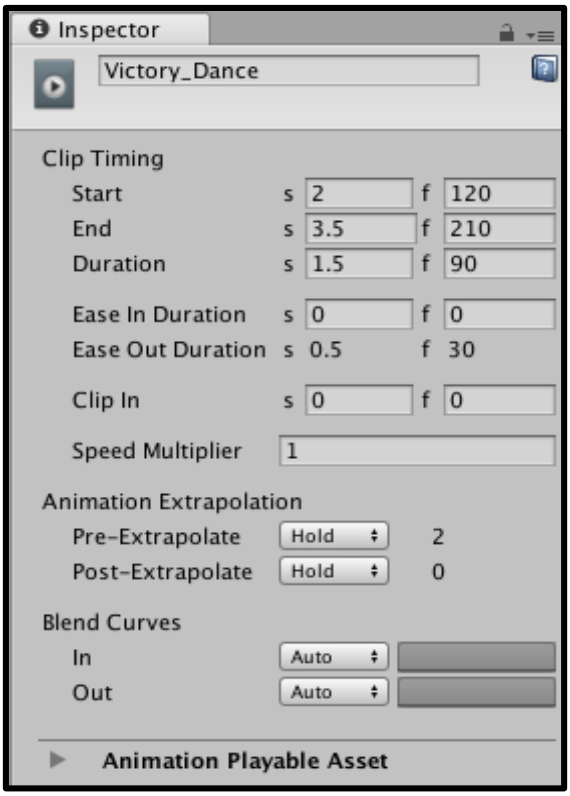


Fig 8.5 Inspector Window

8.7 AUDIO SOURCES AND LISTENERS

A game would be incomplete without some kind of audio, be it background music or sound effects. Unity's audio system is flexible and powerful. It can import most standard audio file formats and has sophisticated features for playing sounds in 3D space, optionally with effects like echo and filtering applied. Unity can also record audio from any available microphone on a user's machine for use during gameplay or for storage and transmission.

In real life, sounds are emitted by objects and heard by listeners. The way a sound is perceived depends on a number of factors. A listener can tell roughly which direction a sound is coming from and may also get some sense of its distance from its loudness and quality. A fast-moving sound source (like a falling bomb or a passing police car) will change in pitch as it moves as a result of the Doppler Effect. Also, the surroundings will affect the way sound is reflected, so a voice inside a cave will have an echo but the same voice in the open air will not as shown in Fig 8.6.

Fig 8.6 Audio Sources and Listener

To simulate the effects of position, Unity requires sounds to originate from Audio Sources attached to objects. The sounds emitted are then picked up by an Audio Listener attached to another object, most often the main camera. Unity can then simulate the effects of a source's distance and position from the listener object and play them to the user accordingly. The relative speed of the source and listener objects can also be used to simulate the Doppler Effect for added realism.

Unity can't calculate echoes purely from scene geometry but you can simulate them by adding Audio Filters to objects. For example, you could apply the Echo filter to a sound that is supposed to be coming from inside a cave. In situations where objects can move in and out of a place with a strong echo, you can add a Reverb Zone to the scene. For example, your game might involve cars driving through a tunnel. If you place a reverb zone inside the tunnel then the cars' engine sounds will start to echo as they enter and the echo will die down as they emerge from the other side.

The Unity Audio Mixer allows you to mix various audio sources, apply effects to them, and perform mastering.

8.7.1 Working with Audio Assets

Unity can import audio files in AIFF, WAV, MP3 and Ogg formats in the same way as other assets, simply by dragging the files into the Project panel. Importing an audio file creates an Audio Clip which can then be dragged to an Audio Source or used from a script. The Audio Clip reference page has more details about the import options available for audio files.

For music, Unity also supports tracker modules, which use short audio samples as “instruments” that are then arranged to play tunes. Tracker modules can be imported from .xm, .mod, .it, and .s3m files but are otherwise used in much the same way as ordinary audio clips.

8.8 INPUT

A game that does not take any input from the user is not much of a game. There are a lot of different kinds of input we can read in, and almost all of them are accessible through the input and key code objects. Some sample input statements are below:

```
Vector3 mousePos = Input.mousePosition;
Bool isLeftClicking = Input.GetMouseButton (0);
```

The functions of these lines is mostly self-explanatory. Using these three kinds of input reference, we can reconstruct the control schemes of most modern 3D computer games.

CHAPTER 9

CONCLUSION

Concerning the managerial aspects, the member acquired experience in how to apply an iterative process for game development. Due to the fact that the group consists of four members, all have the responsibility of all the roles for continuity of the project. Therefore, all team members participated in producing each document. The responsibility of steering the group and keeping the project on track is on all members' hands. The result of the first iteration of the process is the documentation about project planning that is necessary for the successful progress. Because of lack of experience in game development, we faced some difficulties while producing the game design document. All the members of the group must have overall idea of the game concept and mechanics clearly to do implementation. At the beginning, we could not estimate which properties could be implemented within project time. Due to time shortage, we had to put 3D graphics design and creation out of project scope. The testing process of this project went mostly as we had planned; however the test plan schedule was pretty much followed. Because implementation took longer time as we expected, we had to spend less time on testing process but we maintained that too. The end result of the implementation is a game that almost fulfills all of its functional requirements.

REFERENCES

[1] Orland, Kyle "How new graphics effects can make Unity Engine games look less generic"

[2] Hejlsberg, Anders. "Future directions for C# and Visual Basic". C# lead architect. Microsoft

[3] Prudom, Laura "The Flash: Robbie Amell Cast as Firestorm". Variety Archived from the original on July 13

[4] Brodkin, Jon "How Unity3D Became a Game-Development Beast". Dice Insights.

[5] Mullen, T (2009). Mastering Blender. 1st ed. Indianapolis, Indiana: Wiley Publishing, Inc.

[6] Blenderfoundation "Blender 2.58a update log" blender.org